(19)日本国特許庁（ＪＰ）　(12) **公開特許公報**（Ａ）　(11)特許出願公開番号
特開2002－49484
（P2002－49484A）
(43)公開日　平成14年２月15日(2002.2.15)

(51)Int.Cl.[7]	識別記号	ＦＩ		テーマコード（参考）
Ｇ０６Ｆ　9/44		Ｃ０６Ｆ　13/00	５５０Ａ	５Ｂ０７６
13/00	５５０	9/06	６２０Ａ	

審査請求　未請求　請求項の数28　ＯＬ　外国語出願　（全 75 頁）

(21)出願番号　特願2001－129923(P2001－129923)

(22)出願日　平成13年４月26日(2001.4.26)

(31)優先権主張番号　０９／５７３７６９

(32)優先日　平成12年５月18日(2000.5.18)

(33)優先権主張国　米国（ＵＳ）

(71)出願人　595124929
マイクロソフト　コーポレイション
ＭＩＣＲＯＳＯＦＴ　ＣＯＲＰＯＲＡＴＩＯＮ
アメリカ合衆国　98052－6399　ワシントン州　レドマンド　ワン　マイクロソフト　ウェイ　（番地なし）

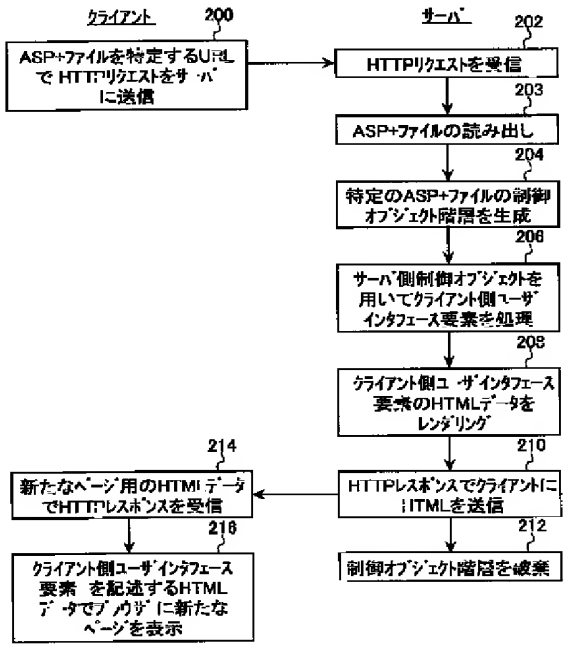
(72)発明者　バード、ゲイリー　エス.
アメリカ合衆国、98033　ワシントン州、カークランド、エヌイー　103ド　ストリート　11411

(74)代理人　100095555
弁理士　池内　寛幸　（外３名）

最終頁に続く

(54)【発明の名称】　クライアント側ユーザインタフェース要素を処理するサーバ側制御オブジェクト

(57)【要約】　（修正有）
【課題】ユーザインタフェース要素を処理することが要求されるプログラミングを適切にカプセル化すること。
【解決手段】サーバ側制御オブジェクトは、ウェブページ上に表示するクライアント側ユーザインタフェース要素を処理し生成するために、多数のサーバ側制御オブジェクトを組み合わせて、クライアント上のウェブページの表示のためのHTMLのような結果オーサリング言語コードを協働してサーバ側制御オブジェクトの階層を生成にする。クライアント側ユーザインタフェース要素の処理は、イベントハンドリング処理、ポストバックデータハンドリング処理、データ結合処理およびサーバ側制御オブジェクトの状態に関する状態管理処理のうちの１つ以上の処理を行う。



【特許請求の範囲】
【請求項１】　クライアント上で表示されるウェブページに組み込まれるクライアント側ユーザインタフェース要素の処理方法であって、
資源を参照するリクエストを受信する工程と、
前記資源から宣言を読み出す工程と、
前記クライアント側ユーザインタフェース要素の機能を提供するように前記宣言に基いてプログラムされたサーバ側制御オブジェクトを生成する工程と、
前記サーバ側制御オブジェクトを用いて、前記クライアント側ユーザインタフェース要素を処理する工程と、
前記処理工程に引き続いて、前記ウェブページに前記クライアント側ユーザインタフェース要素を組み込むためのオーサリング言語データを前記サーバ側制御オブジェクトから生成する工程とを含むクライアント側ユーザインタフェース要素の処理方法。
【請求項２】　前記資源が、サーバ側宣言データ格納部である請求項１に記載のクライアント側ユーザインタフェース要素の処理方法。
【請求項３】　前記オーサリング言語データを生成する工程に引き続いて、前記オーサリング言語データを前記クライアントに送信する工程と、
前記サーバ側制御オブジェクトを終了させる工程とをさらに含む請求項１に記載のクライアント側ユーザインタフェース要素の処理方法。
【請求項４】　前記処理工程は、前記サーバ側制御オブジェクトを用いて、前記ユーザインタフェース要素から受信したポストバックデータをハンドリングする工程を含む請求項１に記載のクライアント側ユーザインタフェース要素の処理方法。
【請求項５】　前記処理工程は、前記サーバ側制御オブジェクトを用いて、前記ユーザインタフェース要素から受信したポストバックイベントをハンドリングする工程を含む請求項１に記載のクライアント側ユーザインタフェース要素の処理方法。
【請求項６】　前記処理工程は、前記サーバ側制御オブジェクトのビューステートを保存する工程と、前記クライアントに前記ビューステートを送信する工程とを含む請求項１に記載のクライアント側ユーザインタフェース要素の処理方法。
【請求項７】　前記処理工程は、
前記クライアントから、前記サーバ側制御オブジェクトのビューステートであって、先のリクエストの前記サーバ側制御オブジェクトの状態に対応するビューステートを受信する工程と、
前記サーバ側制御オブジェクトへ前記ビューステートをロードする工程とを含む請求項１に記載のクライアント側ユーザインタフェース要素の処理方法。
【請求項８】　コンピュータシステムによって搬送波で具体化され、かつクライアント上に表示されるウェブページに組み込まれるクライアント側ユーザインタフェース要素を処理するコンピュータプロセスを実行処理するコンピュータプログラムをコード化しているコンピュータデータ信号であって、
前記コンピュータプロセスは、
資源を参照するリクエストを受信する工程と、
前記資源から宣言を読み出す工程と、
前記クライアント側ユーザインタフェース要素の機能を提供するように前記宣言に基いてプログラムされたサーバ側制御オブジェクトを生成する工程と、
前記サーバ側制御オブジェクトを用いて、前記クライアント側ユーザインタフェース要素を処理する工程と、
前記処理工程に引き続いて、前記ウェブページに前記クライアント側ユーザインタフェース要素を組み込むためのオーサリング言語データを前記サーバ側制御オブジェクトから生成する工程とを含むコンピュータデータ信号。
【請求項９】　コンピュータシステムによって読み取り可能であり、かつ、クライアント上で表示されるウェブページに組み込まれるクライアント側ユーザインタフェース要素を処理するコンピュータプロセスを実行処理するコンピュータプログラムをコード化しているコンピュータプログラム記憶媒体であって、
前記コンピュータプロセスは、
資源を参照するリクエストを受信する工程と、
前記資源から宣言を読み出す工程と、
前記クライアント側ユーザインタフェース要素の機能を提供するように前記宣言に基いてプログラムされたサーバ側制御オブジェクトを生成する工程と、
前記サーバ側制御オブジェクトを用いて、前記クライアント側ユーザインタフェース要素を処理する工程と、
前記処理工程に引き続き、前記ウェブページに前記クライアント側ユーザインタフェース要素を組み込むためのオーサリング言語データを前記サーバ側制御オブジェクトから生成する工程とを含むコンピュータプログラム記憶媒体。
【請求項１０】　クライアント上に表示されるウェブページに組み込まれる１つ以上のクライアント側ユーザインタフェース要素を処理する、コンピュータで実行処理可能なサーバ側制御オブジェクトの階層であって、
前記サーバ側制御オブジェクトの階層は、
前記１つ以上のクライアント側ユーザインタフェース要素に対応する１つ以上のサーバ側子オブジェクトであって、各サーバ側子オブジェクトは、クライアントから受信したポストバック入力をハンドリングするようになっており、かつクライアント上でクライアント側ユーザインタフェース要素を表示するためのオーサリング言語データを生成するようになっている１つ以上のサーバ側子オブジェクトと、
前記サーバ側子オブジェクトのうちの１つを特定してい

るポストバック入力に関連するクライアントから受信した少なくとも１つの階層識別子と、
前記１つ以上のサーバ側子オブジェクトに階層的に関連付けられ、前記階層識別子に基づき前記サーバ側子オブジェクトのうちの１つに分配するポストバック入力を受信するサーバ側ページオブジェクトとを含むサーバ側制御オブジェクトの階層。
【請求項１１】　前記ページオブジェクトが、前記１つ以上のサーバ側子オブジェクトを階層的に含む請求項１０に記載のサーバ側制御オブジェクトの階層。
【請求項１２】　前記ページオブジェクトが、クライアントからのリクエストに応答して作成され、前記ウェブページの前記オーサリング言語データが生成されるまで存在し続ける請求項１０に記載のサーバ側制御オブジェクトの階層。
【請求項１３】　各サーバ側子オブジェクトが、前記階層識別子によって特定されたサーバ側制御オブジェクトへのアクセスに応答して作成され、クライアント上の対応するクライアント側ユーザインタフェース要素の前記オーサリング言語データが生成されるまで存在し続ける請求項１０に記載のサーバ側制御オブジェクトの階層。
【請求項１４】　コンピュータシステムによって読み取り可能であり、かつ、クライアント上で表示されるウェブページに組み込まれる１つ以上のクライアント側ユーザインタフェース要素を処理するコンピュータプロセスを実行処理するコンピュータプログラムをコード化しているコンピュータプログラム記憶媒体であって、
前記コンピュータプロセスは、
サーバ側宣言データ格納部から１つ以上の宣言を入力する工程と、
前記クライアント側ユーザインタフェース要素の機能を提供するように前記宣言に基いてプログラムされたサーバ側制御オブジェクトの階層を生成する工程と、
前記サーバ側制御オブジェクトの階層を用いて、前記クライアント側ユーザインタフェース要素を処理する工程と、
前記ウェブページに前記クライアント側ユーザインタフェース要素を組み込むためのオーサリング言語データを前記サーバ側制御オブジェクトの階層から生成する工程とを含むコンピュータプログラム記憶媒体。
【請求項１５】　前記サーバ側制御オブジェクトの階層の生成工程は、
前記ウェブページ上のクライアント側ユーザインタフェースコンテナ要素に対応するサーバ側コンテナ制御オブジェクトを生成する工程と、
前記クライアント側ユーザインタフェースコンテナ要素に含まれた１つ以上のクライアント側ユーザインタフェース子要素に対応する１つ以上のサーバ側子制御オブジェクトを生成する工程とを含む請求項１４に記載のコンピュータプログラム記憶媒体。
【請求項１６】　前記クライアント側ユーザインタフェース要素の処理工程は、
前記サーバ側制御オブジェクトの階層内の１つ以上のサーバ側制御オブジェクトを参照する一意の階層識別子を受信する工程と、
前記一意の階層識別子に関連するポストバック入力情報を受信する工程と、
参照されたサーバ側制御オブジェクトを識別するために前記一意の階層識別子を分解する工程と、
前記参照されたサーバ側制御オブジェクトに前記ポストバック入力情報を渡す工程と、
前記参照されたサーバ側制御オブジェクトを用いて前記ポストバック入力情報を処理する工程とを含む請求項１５に記載のコンピュータプログラム記憶媒体。
【請求項１７】　前記クライアント側ユーザインタフェース要素の処理工程は、１つ以上のサーバ側制御オブジェクトによる個々の処理を呼び出すために前記サーバ側制御オブジェクトの階層をトラバースする工程を含む請求項１４に記載のコンピュータプログラム記憶媒体。
【請求項１８】　前記クライアント側ユーザインタフェース要素の処理工程は、
前記サーバ側制御オブジェクトのうちの１つによって生成された制御オブジェクトイベントをハンドリングするために１つ以上のサーバ側制御オブジェクトを登録する工程と、
１つ以上の前記サーバ側制御オブジェクトから前記制御オブジェクトイベントを生成する工程と、
前記制御オブジェクトイベントをハンドリングするために登録された１つ以上の前記サーバ側制御オブジェクトを用いて前記制御オブジェクトイベントをハンドリングする工程とを含む請求項１４に記載のコンピュータプログラム記憶媒体。
【請求項１９】　前記クライアント側ユーザインタフェース要素の処理工程は、
１つ以上の前記サーバ側制御オブジェクトに第１のビューステートをロードする工程と、
前記第１のビューステートのロード工程に引き続き、１つ以上の前記サーバ側制御オブジェクトを用いてポストバックデータをハンドリングする工程と、
前記ポストバックデータをハンドリングする工程に引き続き、１つ以上の前記サーバ側制御オブジェクトを用いてポストバックイベントをハンドリングする工程と、
前記ポストバックイベントをハンドリングする工程に引き続き、１つ以上の前記サーバ側制御オブジェクトからの第２のビューステートを保存する工程とを含む請求項１４に記載のコンピュータプログラム記憶媒体。
【請求項２０】　前記クライアント側ユーザインタフェース要素の処理工程は、
ポストバックイベントをハンドリングする工程に引き続き、サーバ側制御オブジェクトとサーバ側データ格納部

との間のデータ結合関係を解明する工程をさらに含む請求項１９に記載のコンピュータプログラム記憶媒体。
【請求項２１】　クライアント上に表示されるウェブページに組み込まれる少なくとも１つのクライアント側ユーザインタフェース要素の処理方法であって、
資源から宣言を読み出す工程と、
前記宣言に基づいて、前記クライアント側ユーザインタフェース要素と論理的に対応する複数の同時存在するサーバ側制御オブジェクトを生成する工程と、
前記同時存在するサーバ側制御オブジェクトを用いて、クライアント側ユーザインタフェース要素を処理する工程と、
前記処理工程に引き続き、前記ウェブページに前記クライアント側ユーザインタフェース要素を組み込むためのオーサリング言語データを前記同時存在するサーバ側制御オブジェクトから生成する工程とを含むクライアント側ユーザインタフェース要素の処理方法。
【請求項２２】　前記複数の同時存在するサーバ側制御オブジェクトの生成工程は、
第１のクライアント側ユーザインタフェース要素に対応する第１のサーバ側制御オブジェクトを生成する工程と、
前記第１のサーバ側制御オブジェクトと前記第２のサーバ側制御オブジェクトが同時に存在するように、第１のクライアント側ユーザインタフェース要素に対応する第２のサーバ側制御オブジェクトを生成する工程とを含む請求項２１に記載のクライアント側ユーザインタフェース要素の処理方法。
【請求項２３】　前記処理工程が、第１のサーバ側制御オブジェクトに関連するサーバ側イベントを立ち上げる工程を含む請求項２１に記載のクライアント側ユーザインタフェース要素の処理方法。
【請求項２４】　前記処理工程が、第２のサーバ側制御オブジェクトを用いて前記サーバ側イベントをハンドリングする工程をさらに含む請求項２３に記載のクライアント側ユーザインタフェース要素の処理方法。
【請求項２５】　前記処理工程が、非ユーザインタフェースサーバコンポーネントを用いてサーバ側イベントをハンドリングする工程をさらに含む請求項２３に記載のクライアント側ユーザインタフェース要素の処理方法。
【請求項２６】　前記複数の同時存在するサーバ側制御オブジェクトの生成工程が、
クライアント側ユーザインタフェース要素に対応する第１のサーバ側制御オブジェクトを生成する工程と、
前記第１のサーバ側制御オブジェクトと前記第２のサーバ側制御オブジェクトが同時に存在するように前記クライアント側ユーザインタフェース要素に対応する第２のサーバ側制御オブジェクトを生成する工程とを含む請求項２１に記載のクライアント側ユーザインタフェース要素の処理方法。
【請求項２７】　コンピュータシステムによって搬送波で具体化され、クライアント上で表示されるウェブページに組み込まれる少なくとも１つのクライアント側ユーザインタフェース要素を処理するコンピュータプロセスを実行処理するコンピュータプログラムをコード化しているコンピュータデータ信号であって、
前記コンピュータプロセスは、
宣言を資源から読み出す工程と、
前記宣言に基づいて、前記クライアント側ユーザインタフェース要素と論理的に対応する複数の同時存在するサーバ側制御オブジェクトを生成する工程と、
前記同時存在するサーバ側制御オブジェクトを用いて、クライアント側ユーザインタフェース要素を処理する工程と、
前記処理工程に引き続き、前記ウェブページに前記クライアント側ユーザインタフェース要素を組み込むためのオーサリング言語データを前記同時存在するサーバ側制御オブジェクトから生成する工程とを含むコンピュータデータ信号。
【請求項２８】　コンピュータシステムによって読み取り可能であり、かつ、クライアント上で表示されるウェブページに組み込まれる少なくとも１つのクライアント側ユーザインタフェース要素を処理するコンピュータプロセスを実行処理するコンピュータプログラムをコード化しているコンピュータプログラム記憶媒体であって、
前記コンピュータプロセスは、
宣言を資源から読み出す工程と、
前記宣言に基づいて、前記クライアント側ユーザインタフェース要素と論理的に対応する複数の同時存在するサーバ側制御オブジェクトを生成する工程と、
前記同時存在するサーバ側制御オブジェクトを用いて、クライアント側ユーザインタフェース要素を処理する工程と、
前記処理工程に引き続き、前記ウェブページに前記クライアント側ユーザインタフェース要素を組み込むためのオーサリング言語データを前記同時存在するサーバ側制御オブジェクトから生成する工程とを含むコンピュータプログラム記憶媒体。
【発明の詳細な説明】
【０００１】
【発明の属する技術分野】本発明は、一般にウェブサーバフレームワークに関し、特にウェブページのクライアント側ユーザインタフェース要素を処理するサーバ側制御オブジェクトに関する。
【０００２】
【従来の技術】典型的なウェブブラウザは、クライアントシステムにおいて表示するウェブページの外観および基本動作を定義するウェブサーバからデータを受け取る。典型的な手順としては、ユーザが、ワールドワイドウェブ上の資源のグローバルアドレスであるユニホーム

リソース（資源）ロケータ（以下、「URL」という。）を特定し、所望のウェブサイトにアクセスする。一般に、用語「資源」とは、プログラムによってアクセスすることができるデータまたはルーチンのことである。

【０００３】URLの一例は、“ HYPERLINK ”http://www.microsoft.com/ms.htm” http://www.microsoft.com/ms.htm ”である。このURL例の第１の部分は、通信に用いる所与のプロトコル（例えば“http”）を示している。第２の部分は、資源の所在を示すドメイン名（例えば“ HYPERLINK ”http://www.microsoft.com” www.microsoft.com ”）を特定している。第３の部分はドメイン内の資源（例えば“ms.htm”と呼ばれるファイル）を特定している。これに従い、ブラウザは、 HYPERLINK”http://www.microsoft.com” www.microsoft.com ドメイン内のms.htmファイルに関連するデータを取り出すためのURL例に関連するHTTP(ハイパーテキストトランスポートプロトコル)リクエストを生成する。www.microsoft.comサイトをホストしているウェブサーバはHTTPリクエストを受信し、要求されたウェブページまたは資源をクライアントシステムにHTTPレスポンスで戻し、ブラウザに表示する。

【０００４】上記の例の”ms.htm”ファイルは、静的なHTML（ハイパーテキストマークアップ言語）コードを含んでいる。HTMLは、ワールドワイドウェブ上でのドキュメント（例えば、ウェブページ）作成に用いられるプレイン（平文）テキストオーサリング言語である。このようなものであるので、HTMLファイルは、ウェブサーバから取り出され、インターネットからの情報を表示している間、ユーザが期待するグラフィカルな体験を提供するために、ブラウザにウェブページとして表示することができる。

【０００５】HTMLを用いて、開発者は、例えば、ブラウザに表示するフォーマット化されたテキスト、リスト、フォーム、テーブル ハイパーテキストリンク、インライン画像および音声、ならびに背景グラフィックスを特定することができる。しかし、HTMLファイルは、ウェブページコンテンツの動的な生成を本質的にサポートしていない静的ファイルである。

【０００６】また、ウェブページは、ブラウザに株価の変化や交通情報など動的コンテンツを表示する必要がある場合がある。このような状況では、典型的なサーバ側アプリケーションプログラムは、動的データを取得し、それを、ウェブページが更新されるとともにウェブページに表示するブラウザに送信されるHTML形式にフォーマットするように開発される。

【０００７】さらに、データは厳密には動的でないが、静的なウェブページに表示する値が非常に多くの異なる値であるために、要求された数の静的なウェブページを作成することが実用的でないような状況において、これらの同様のサーバ側アプリケーションプログラムを用いることができる。例えば、旅行計画ページは、出発日用カレンダーと帰着日用カレンダーとの２種類のカレンダー表示をすることがある。考え得るすべてのカレンダーの組み合わせによる何百もの静的なページを開発するかわりに、サーバ側アプリケーションプログラムは、適切なカレンダーを表示した適切な静的なページを動的に生成することができる。

【０００８】多くのウェブページにより、ユーザはページの視覚的要素を選択することによってブラウザに表示されたページと対話することができる。例えば、上記の旅行計画ページにおいて、カレンダーによって、ユーザは日付をクリックしてその日付を選ぶ、またはアイコンをクリックして月を進めたり戻ったりすることによってそのカレンダーと対話することができる。既存の解決法では、ブラウザは、サーバ側アプリケーションプログラムにHTTPリクエストを出す。このHTTPリクエストは、フォームポスト変数のような照会行列あるいはその他、クライアント側イベントまたはデータ(例えば、どのコントロールをユーザがクリックしたか)を記述するデータフォーマットでコード化されたパラメータを含むことができる。例えば、パラメータは、１つのカレンダーでユーザが選択したデータを他方のカレンダーで現在表示中のデータとともに含む場合がある。

【０００９】サーバに戻すイベントおよびデータの通信は、「ポストバック」と呼ばれる。これは、典型的に、ブラウザが、HTTPポストリクエストを用いてリクエストを送信するからである。サーバ側アプリケーションプログラムは、HTTPリクエストを処理し、クライアントへHTTPレスポンスで送信するユーザのアクションを反映する新たに計算されたカレンダーを有するウェブページに適切なHTMLコードを生成する。その後、得られたドキュメントは、HTTPレスポンスでクライアントシステムに送信され、このドキュメントは、更新されたカレンダーを示すウェブページとしてブラウザに表示される。

【００１０】サーバ側アプリケーションプログラムの開発は、ウェブページ設計に用いる通常のHTMLコード化に精通しているだけでなく、どのようにデータがブラウザとサーバとの間で送信されるか、または１つ以上のプログラム言語（例えば、C++, Perl, Visual Basic, または Jscript）およびHTTPプロトコルを含むプログラムベーシックスに精通していることが要求される複雑な作業である。しかし、ウェブページ設計者は、グラフィックデザイナーまたはエディタであることが多く、プログラム経験がない場合がある。さらに、複雑なウェブページ開発を単純化すると、いかなる開発者によっても新たなウェブコンテンツの開発の速度を上げることができる。

【００１１】一般に、カスタムサーバ側アプリケーションプログラムの開発もまた、多大な努力が要求され、実際、開発者はしばしばそれを試みたくないと思う程である。開発者は、所望のウェブページを表示するために生

成が要求されるHTMLコードを理解するだけでなく、ウェブページからのユーザ対話およびクライアントデータがどのようにしてポストバック処理になるのかを理解しなければならない。したがって、開発者が最小のプログラミングでウェブページを動的に作成および処理することができる開発フレームワークを提供することが望ましい。

【００１２】

【発明が解決しようとする課題】動的ウェブページ生成のプログラム要件を最小にする１つの手段は、マイクロソフト社によって提供されるアクティブサーバページ（ASP）フレームワークである。ASPリソースは、典型的に、所望の資源としてASPリソースを特定するHTTPリクエストを処理し、その後、クライアントへのHTTPレスポンスでの結果HTMLコードを生成する、例えばVisual BasicまたはJscriptを含む。さらに、ASP資源は、所与のアプリケーションプログラミング努力を緩和するために、予め開発されたまたは第三者のクライアント側ライブラリコンポーネント（例えば、クライアント側"ACTIVEX"制御）を参照することができる。しかし、現在のサーバ側アプリケーションフレームワークにおいて、サーバ側アプリケーション内でクライアント側ユーザインタフェース要素（例えば、テキストボックス、リストボックス、ボタン、ハイパーリンク、画像、音声など）を動的に管理しなければならないプログラミングは、依然として高度なプログラミング技術と相当な努力を必要とする。未解決の課題は、ウェブページ開発者がウェブページの他のアスペクトに焦点を当てることができるように、ポストバックイベントをハンドリングすることなど、ユーザインタフェース要素を処理することが要求されるプログラミングを適切にカプセル化することについてである。

【００１３】

【課題を解決するための手段】本発明によると、上述および他の課題は、クライアント側ユーザインタフェース要素の処理および生成を管理するサーバ側制御オブジェクトフレームワークを提供することによって解決される。さらに、サーバ側制御オブジェクトの階層が、クライアントにウェブページを表示するための標準HTMLのような結果オーサリング言語コードを協働して生成することができる。クライアントは、例えば標準HTMLまたは別の結果オーサリング言語コードをサポートするものであればいかなるブラウザであってもよい。クライアント側ユーザインタフェース要素の処理は、１つ以上のポストバックイベントハンドリング処理、ポストバックデータハンドリング処理、データ結合処理またはサーバ側制御オブジェクトの状態に関する状態管理処理を含む。

【００１４】本発明のある実施形態の大きな有効性は、クライアント側ユーザインタフェース要素（例えば、ユーザインタフェース要素からの受信入力およびユーザインタフェース要素の生成に用いる出力）および関連機能性のサーバ側処理のカプセル化の向上にある。１つ以上のサーバ側制御オブジェクトは、１つ以上のユーザインタフェース要素に論理的に対応するように生成される場合がある。例えば、カレンダー表示において月を表すユーザインタフェース要素を考えると、サーバ側制御オブジェクトの階層は、カレンダー表示およびその種々のサブ要素に対応して生成される場合がある。ある構成では、「月」制御オブジェクトは、多数の「週」制御オブジェクトを階層的に含み、各「週」制御オブジェクトは、７つの「日」制御オブジェクトを階層的に含んでいる。

【００１５】さらに、本発明のある実施形態において、サーバ側制御オブジェクトは、論理的に対応するユーザインタフェース要素を協働して処理することができる。この利点は、クライアントリクエスト処理およびレスポンスの生成中において多数のサーバ側制御オブジェクトが同時に存在することなどによりもたらされた結果である。例えば、クライアントから受信した第１のポストバックイベント（例えば、カレンダー表示上の「次の月」のボタン要素のクリック）を検知すると、１つの制御オブジェクトは、その後に検知され１つ以上の同時存在している制御オブジェクトによって処理される第２のイベント（例えば、「月」制御オブジェクトによる次の月の選択および表示）を立ち上げる。この協働により、サーバ側制御オブジェクト自体内で複雑な制御対話のカプセル化を行うことができ、これにより、ウェブページ開発者に要求されるカスタムイベントハンドリングを最小限にすることができる。

【００１６】クライアント上で表示されるウェブページに組み込まれるクライアント側ユーザインタフェース要素を処理する方法およびコンピュータプログラムプロダクトが提供される。サーバ側資源データ格納部を参照するリクエストを受信する。宣言が、サーバ側宣言データ格納部から入力される。サーバ側制御オブジェクトが、宣言に基いてユーザインタフェース要素の機能性を与えるように生成およびプログラムされる。サーバ側制御オブジェクトを用いて、ユーザインタフェース要素が処理される。オーサリング言語データが、ウェブページにユーザインタフェース要素を表示するサーバ側制御オブジェクトから生成される。

【００１７】クライアント上に表示されるウェブページに組み込まれる１つ以上のクライアント側ユーザインタフェース要素を処理する、コンピュータで実行処理可能なサーバ側制御オブジェクトの階層が提供される。１つ以上のサーバ側子オブジェクトは、１つ以上のクライアント側ユーザインタフェース要素に対応する。各サーバ側子オブジェクトは、クライアントから受信した入力を扱い、クライアント上でクライアント側ユーザインタフェース要素を表示するためのオーサリング言語データを

生成する。少なくとも１つの階層識別子が、サーバ側子オブジェクトのうちの１つを特定している入力に関連するクライアントから受信される。サーバ側ページオブジェクトは、サーバ側子オブジェクトを含み、階層識別子に応じたサーバ側子オブジェクトのうちの１つへ分配する入力を受信する。

【００１８】

【発明の実施の形態】本発明のある実施形態は、ウェブページ上に表示するクライアント側ユーザインタフェース要素を処理および生成するサーバ側制御オブジェクトを含む。さらに、サーバ側制御オブジェクトの階層が、クライアントにウェブページを表示するための標準HTMLのような結果オーサリング言語コードを協働して生成することができる。クライアントは、例えば標準HTMLまたは別の結果オーサリング言語コードをサポートするものであればいかなるブラウザであってもよい。本発明のある実施形態では、サーバ側制御オブジェクトは、クライアント側ユーザインタフェース要素と論理的に対応し、ウェブページを表示および処理するクライアント側ブラウザによって用いられるオーサリング言語コードをサーバで生成する。クライアント側ユーザインタフェース要素の処理は、１つ以上のポストバックイベントハンドリング処理、ポストバックデータハンドリング処理、データ結合処理および状態管理処理を含んでも良い。

【００１９】図１は、本発明のある実施形態におけるクライアントに表示するウェブページコンテンツを動的に生成するウェブサーバを示す。クライアント１００は、クライアント１００の表示装置上におけるウェブページ１０４を表示するブラウザ１０２を実行する。クライアント１００は、ビデオモニタのような表示装置を有するクライアントコンピュータシステムを含んでもよい。マイクロソフト社によって販売されている"INTERNET EXPLORER"ブラウザは、本発明のある実施形態におけるブラウザ１０２の一例である。他のブラウザの例として、"NETSCAPE NAVIGATOR" および "MOSAIC"などがあるが、これに限らない。例示したウェブページ１０４には、テキストボックス制御１０６および２つのボタン制御１０８、１１０が組み込まれている。ブラウザ１０２は、HTTPレスポンス１１２でウェブサーバ１１６からHTMLコードを受け取ることができ、HTMLコードにより記述されたウェブページを表示する。ある実施形態を参照してHTMLについて説明するが、特に制限されるものではなく、SGML（Standard Generalized Markup Language）、XML（eXtensible Markup Language）および、XMLベースのマークアップ言語であって、ポケットベル（登録商標）および携帯電話などの狭帯域無線装置の内容およびユーザインタフェースを特定するように設計されたWML（Wireless Markup Language）を含む他のオーサリング言語も本発明の範囲内であると考えられている。さらに、標準HTML３．２が、本明細書中に主に開示されているが、HTMLのいかなるバージョンも本発明の範囲内に含まれ得る。

【００２０】クライアント１００とウェブサーバ１１６との通信は、HTTPリクエスト１１４およびHTTPレスポンス１１２の一連の処理を用いて行うことができる。ある実施形態を参照してHTTPを説明するが、特に制限されるものではなく、S－HTTPを含めた他のトランスポートプロトコルも本発明の範囲内であると考えられている。ウェブサーバ１１６において、HTTPパイプラインモジュール１１８が、HTTPリクエスト１１４を受信し、URLを解析し、リクエストを処理する適切なハンドラを呼び出す。本発明のある実施形態において、異なるタイプの資源をハンドリングする複数のハンドラ１２０がウェブサーバ１１６に備わっている。

【００２１】例えば、URLがHTMLファイルのような静的なコンテンツ資源１２２を特定する場合、ハンドラ１２０は、静的コンテンツ資源１２２にアクセスし、HTTPパイプライン１１８を介して静的コンテンツ資源１２２をHTTPレスポンス１１２でクライアント１００へ送る。あるいは、本発明のある実施形態において、URLがASP+リソースのような動的コンテンツ資源１２４を特定する場合、ハンドラ１２０は、動的コンテンツ資源１２４にアクセスし、動的コンテンツ資源１２４のコンテンツを処理し、ウェブページ１０４用の結果HTMLコードを生成する。本発明のある実施形態において、結果HTMLコードは、標準HTML３．２コードを含む。一般に、動的コンテンツ資源は、クライアントに表示すべきウェブページを記述するオーサリング言語を動的に生成するのに用いることができるサーバ側宣言データ格納部（例えば、ASP+リソース）である。そして、ウェブページ用のHTMLコードは、HTTPパイプライン１１８を通過し、HTTPレスポンス１１２でクライアント１００へ送られる。

【００２２】この処理の間、ハンドラ１２０はまた、開発労力を簡単化するために予め開発された、あるいは第３者コードのライブラリにアクセスすることができる。このようなライブラリの１つがサーバ側クラス制御ライブラリ１２６であり、ここから、ハンドラ１２０は、ユーザインタフェース要素を処理しウェブページに表示する結果HTMLデータを生成するサーバ側制御オブジェクトを例示することができる。本発明のある実施形態において、１つ以上のサーバ側制御オブジェクトは、動的コンテンツ資源１２４に記述されたウェブページ上に、可視的にまたは隠して、１つ以上のユーザインタフェース要素にマッピングする。

【００２３】これに対し、第２のライブラリは、マイクロソフト社からの"ACTIVEX"コンポーネントを含むライブラリのようなクライアント側制御クラスライブラリ１２８である。"ACTIVEX"制御は、クライアントおよび他のコンポーネントとの対話の仕方において一定の標準に従うCOM（コンポーネントオブジェクトモデル）オブジェクトである。クライアント側"ACTIVEX"制御は、例え

ば、クライアントに自動的にダウンロードされ、クライアントのウェブブラウザによって実行処理され得るCOMベースのコンポーネントである。サーバ側ACTIVEXコンポーネント（図示せず）は、株価検索アプリケーションまたはデータベースコンポーネントのサーバ側機能性を提供するような多様なサーバ側機能を果たすためにサーバ上で実行され得るCOMベースのコンポーネントである。ACTIVEXについては、「ACTIVEXおよびOLEの理解」（デビッド・チャペル、マイクロソフトプレス、１９９６年）により詳細に記載されている。

【００２４】"ACTIVEX"制御とは対照的に、動的コンテンツ資源１２４に特定される本発明の実施形態のサーバ側制御オブジェクトは、クライアント上の表示されるユーザインタフェース要素に論理的に対応する。サーバ側制御オブジェクトはまた、例えば、HTMLタグと所与のクライアント側"ACTIVEX"制御を参照するロケータを含み得る有効なHTMLコードを生成することができる。ブラウザが、すでに格納システム内にクライアント側"ACTIVEX"制御用コードを有している場合は、クライアント上のウェブページ内で"ACTIVEX"制御を実行処理する。そうでなければ、ブラウザは、ロケータによって特定された資源から"ACTIVEX"制御用コードをダウンロードし、そして、クライアント上のウェブページ内で"ACTIVEX"制御を実行処理する。本発明の実施形態におけるサーバ側制御オブジェクトはまた、サーバ上で株価検索アプリケーションを実行するのに用いられるサーバ側"ACTIVEX"オブジェクトに対しイベントを立ち上げることができる。

【００２５】ハンドラ１２０はまた、ウェブサーバ１１６上または別のアクセス可能ウェブサーバ上で実行処理する１つ以上の非ユーザインタフェースサーバコンポーネント１３０にアクセスする。株価検索アプリケーションまたはデータベースコンポーネントのような非ユーザインタフェースサーバコンポーネント１３０は、ハンドラ１２０によって処理される動的コンテンツ資源１２４において参照され、またはそれに関連付けられる。動的コンテンツ資源１２４で宣言された制御オブジェクトによって立ち上げられたサーバ側イベントは、非ユーザインタフェースサーバコンポーネント１３０の適切なメソッドを呼び出すサーバ側コードによって処理され得る。その結果、サーバ側制御オブジェクトによって提供される処理は、ウェブページのユーザインタフェース要素の処理および生成をカプセル化することによって非ユーザインタフェースサーバコンポーネント１３０のプログラミングを簡単にし、これにより、非ユーザインタフェースサーバコンポーネント１３０の開発者は、ユーザインタフェース問題ではなく、アプリケーション固有の機能の開発に集中することができる。

【００２６】図２は、本発明のある実施形態におけるサーバ側制御オブジェクトを用いてのクライアント側ユーザインタフェース要素の処理および生成処理のフローチャートを示す。処理２００において、クライアントは、HTTPリクエストをサーバに送信する。HTTPリクエストは、ASP+リソースなどの資源を特定するURLを含む。処理２０２において、サーバは、HTTPリクエストを受信し、特定された資源を処理する適切なハンドラを呼び出す。ASP+リソースは処理２０３において読み出される。処理２０４は、特定された動的コンテンツ資源（例えば、ASP+リソース）の内容に基づきサーバ側制御オブジェクト階層を生成する。

【００２７】処理２０６において、制御オブジェクト階層のサーバ側制御オブジェクトは、ポストバックイベントハンドリング、ポストバックデータハンドリング、状態管理およびデータ結合のうちの１つ以上の処理を行う。ユーザインタフェース要素からのポストバックイベントおよびデータ（まとめて「ポストバック入力」）は、クライアントからサーバへと送られ処理される。ポストバックイベントは、特に制限されるものではなく、例えば、クライアント側ボタン要素からの「マウスクリック」またはサーバに送られるクライアント側テキストボックス要素からの「データ変化」イベントを含んでもよい。ポストバックデータは、特に制限されるものではなくが、例えば、テキストボックス要素またはドロップダウンボックスから選ばれた項目のインデックスにユーザによって入力されたテキストを含んでもよい。しかし、ポストバック処理は、他のイベントによって実行されるものでも良く、単にユーザとの対話によって実行されるものとは限らない。

【００２８】処理２０８において、階層内の各サーバ側制御オブジェクトが、クライアント側ユーザインタフェース要素のウェブページでの表示のためのHTMLコードのようなオーサリング言語データを生成（またはレンダリング）するために呼び出される。用語「レンダリング」は、ユーザインタフェース上にグラフィックスを表示する処理を意味することがあるが、本明細書において、用語「レンダリング」は、表示およびクライアント側機能のためのブラウザのようなクライアントアプリケーションによって解釈され得るオーサリング言語データの生成処理をも意味している。処理２０６およびレンダリング処理２０８のより詳細な説明は図６との関連で行われている。ある実施形態において、個々の制御オブジェクトにおけるrender（）メソッドの呼び出しは、ツリートラバーサルシーケンスを用いて行われる。すなわち、ページオブジェクトのrender（）メソッドの呼び出しは、階層内の適切なサーバ側制御オブジェクトにわたる反復トラバースになる。適切な制御オブジェクトのrender（）メソッドを呼び出す別の方法として、イベントシグナリングまたはオブジェクト登録手法などの方法を用いてもよい。括弧は、データ値と比較するなどのメソッドを示す「render（）」レベルを指定する。

【００２９】本発明のある実施形態において、個々のサ

ーバ側制御オブジェクトの実際の作成は、サーバ側制御オブジェクトが処理２０６または２０８においてアクセスされる（ポストバック入力のハンドリング、状態のロード、制御オブジェクトからHTMLコードのレンダリングなど）まで遅らせてもよい。サーバ側制御オブジェクトが所与のリクエストのためにアクセスされることがない場合、制御オブジェクトの作成を遅らせ、不要な制御オブジェクト作成処理を排除することによって、サーバ処理が最適化される。

【００３０】処理２１０において、HTMLコードをHTTPレスポンスでクライアントに送信する。処理２１４において、クライアントは、表示されるべき新たなウェブページに関連するHTMLコードを受信する。処理２１６において、クライアントシステムは、HTTPレスポンスから受け取ったHTMLコードに応じて新しいページのユーザインタフェース要素を組み入れる（例えば、表示する）。しかし、ユーザインタフェース要素の組み入れは、音声または触覚出力を提供する、メモリへの読出しおよび書込み、スクリプト処理の制御などの非表示処理を含んでもよいことを理解すべきである。処理２１２において、サーバ側制御オブジェクト階層を終了する。本発明のある実施形態において、階層内サーバ側制御オブジェクトは、関連のASP+レスポンスを参照するHTTPリクエストに応答して作成され、オーサリング言語データ（例えば、HTMLデータ）のレンダリングが終わると破壊される。別の実施形態において、処理２１２は、処理２０８の後、処理２１０の前に行ってもよい。

【００３１】図３は、本発明のある実施形態において用いるウェブサーバでのモジュールの一例を示す。ウェブサーバ３００は、HTTPパイプライン３０４にHTTPリクエスト３０２を受け入れる。HTTPパイプライン３０４は、ウェブページ統計のロギング、ユーザ照合、ユーザアクセス権およびウェブページの出力キャッシュ化用モジュールなどの種々のモジュールを含んでもよい。ウェブサーバ３００によって受信される各入力HTTPリクエスト３０２は、最終的には、IHTTPハンドラクラス（ハンドラ３０６として図示）の特定のインスタンスによって処理される。ハンドラ３０６は、URLリクエストを分解し適切なハンドラファクトリ（例えば、ページファクトリモジュール３０８）を呼び出す。

【００３２】図３において、ASP+リソース３１０に関連付けられたページファクトリ３０８が呼び出され、ASP+リソース３１０の例示および構成をハンドリングする。ある実施形態において、ASP+リソースは、ファイルに特定の接尾語（または".aspx"のようなファイル拡張部分）を指定することによって認識され得る。所与の".aspx"資源に対するリクエストがまず、ページファクトリモジュール３０８によって受信されると、ページファクトリモジュール３０８は、ファイルシステムを検索して適切なファイル（例えば、.aspxファイル３１０）を得る。このファイルは、リクエストを処理するサーバによって後に解釈またはアクセスされ得るテキスト（例えば、オーサリング言語データ）または別のフォーマットでのデータ（例えば、バイトコードデータまたはコード化されたデータ）を含んでもよい。物理的なファイルが存在する場合、ページファクトリモジュール３０８は、ファイルを開き、そのファイルをメモリに読み出す。ファイルが見つからない場合、ページファクトリモジュール３０８は、適切な「ファイルが見つからない」というエラーメッセージを戻す。

【００３３】ASP+リソース３１０をメモリに読み出した後、ページファクトリモジュール３０８は、ファイル内容を処理し、ページのデータモデル（例えば、スクリプトブロックのリスト、ディレクティブ、静的テキスト領域、階層サーバ側制御オブジェクト、サーバ側制御プロパティなど）を構築する。データモデルは、ページベースのクラスを拡張させるCOM+（Component Object Model+）クラスのような新たなオブジェクトクラスのソースリストを生成するのに用いられる。ページベースクラスは、基本ページオブジェクトの構造、プロパティおよび機能を規定するコードを含んでいる。本発明のある実施形態において、次に、ソースリストは、中間言語に動的にコンパイルされ、のちにプラットフォームに固有の命令（例えば、X86、Alphaなど）に適時に（Just-In-Time）コンパイルされる。中間言語は、COM+ IL コード、Java バイトコード、Modula 3 コード、SmallTalk コードおよびVisual Basicコードなどの汎用またはカスタム指向言語コードを含んでもよい。別の実施形態において、中間言語処理を省き、ネイティブ命令をソースリストまたはソースファイル（例えば、ASP+リソース３１０）から直接生成してもよい。制御クラスライブラリ３１２は、制御オブジェクト階層の生成に用いられる予め定義されたサーバ側制御クラスを得るためにページファクトリモジュール３０８によってアクセスされ得る。

【００３４】ページファクトリモジュール３０８は、図１のウェブページ１０４に相当するサーバ側制御オブジェクトであるページオブジェクト３１４を出力する。ページオブジェクト３１４およびその子オブジェクト（例えば、テキストボックスオブジェクト３１８、ボタンオブジェクト３２０および別のボタンオブジェクト３２２）が、制御オブジェクト階層３１６の一例である。他の制御オブジェクトの例は、本発明に従い考えることができ、カスタム制御オブジェクトと同様に、特に制限されるものではなく、表１のHTML制御に対応するオブジェクトも含んでいる。ページオブジェクト３１４は、図１のウェブページ１０４に対応する。テキストボックスオブジェクト３１８は、図１のテキストボックス１０６に対応する。同様に、ボタンオブジェクト３２０は、図１の追加ボタン１０８に対応し、ボタンオブジェクト３２２は、図１の削除ボタン１１０に対応する。ページオブ

ジェクト３１４は、サーバ上の他の制御オブジェクトと階層的に関連している。ある実施形態において、ページオブジェクトは、その子制御オブジェクトを階層的に含んでいるコンテナオブジェクトである。別の実施形態では、依存関係などの他の形態の階層関係を用いることができる。多数レベルの子オブジェクトを有するより複雑な制御オブジェクト階層において、１つの子オブジェクトが、他の子オブジェクトのコンテナオブジェクトであってもよい。

【００３５】上記の実施形態において、制御オブジェクト階層３１６の制御オブジェクトは、サーバ３００上で作成および実行され、各サーバ側制御オブジェクトは、クライアント上の対応するユーザインタフェース要素と論理的に対応する。サーバ側制御オブジェクトはまた、協働して、HTTPリクエスト３０２からのポストバック入力をハンドリングし、サーバ側制御オブジェクトの状態を管理し、サーバ側データベースとのデータ結合を行い、クライアントでの結果ウェブページの表示に用いるオーサリング言語データ（例えば、HTMLコード）を生成する。結果オーサリング言語データは、サーバ側制御オブジェクト階層３１６から生成（すなわち、レンダリング）され、HTTPレスポンス３２４でクライアントに送信される。例えば、結果HTMLコードは、いかなる有効なHTML構成も具体化することができ、ＡＣＴＩＶＥＸ（登録商標）タイプの制御、ＪＡＶＡ（登録商標）アプレット、スクリプトおよびその他、ブラウザによって処理されたとき、クライアント側ユーザインタフェース要素（例えば、制御ボタン、テキストボックスなど）を生み出すいかなるウェブ資源も参照することができる。

【００３６】ASP+リソース３１０でなされた宣言によって、サーバ側制御オブジェクトは、非ユーザインタフェースサーバコンポーネント３３０とクライアント側ユーザインタフェース要素との対話のために、１つ以上の非ユーザインタフェースサーバコンポーネント３３０にアクセスすることができる。例えば、ポストバック入力に応答して、サーバ側制御オブジェクトは、サーバ側イベントをそれらのイベント用に登録された非ユーザインタフェースサーバコンポーネントに対し立ち上げることができる。このように、非ユーザインタフェースサーバコンポーネント３３０は、ユーザとの対話を、ユーザインタフェース要素を介して、これらの要素を表示および処理するのに必要なコードをプログラミングすることなく行うことができる。

【００３７】図４は、本発明のある実施形態における動的コンテンツ資源の一例の内容を示す。例示された実施例において、ファイル４００は、ある動的コンテンツ資源フォーマット（例えば、ASP+）でプレインテキスト宣言を含んでいる。各宣言は、ファイル４００を読み出すページコンパイラに対し命令を与え、クライアント側ユーザインタフェース要素を処理するために適切なサーバ側制御オブジェクトの作成および呼び出しを行い、最終的にはクライアントへHTTPレスポンスで送信するレンダリングされたHTMLコードを組み合わせる。そして、宣言は、サーバ側制御オブジェクトによって実行されるクライアント側ユーザインタフェース要素の機能を記述または参照する。そして、サーバ側制御オブジェクトは、クライアント上のウェブページの新たなバージョンを定義するために用いるHTMLコードを生成する。

【００３８】ファイル４００の第１ラインは、以下のフォーマットのディレクティブを含む：

<%@ directive [attribute=value] %>

ここで、directiveは、特に制限されるものではなく、"page"、"cache"または"import"を含んでもよい。ディレクティブは、バッファリングセマンテックス、セッション状態要件、エラーハンドリングスキーム、スクリプティング言語、トランザクションセマンテックスおよびインポートディレクティブのような特性を決定するために、動的コンテンツ資源を処理するときにページコンパイラによって用いられる。ディレクティブは、ページファイル内のどこに存在してもよい。

【００３９】第２ラインの<html>は、リテラルとして結果HTMLコードへ渡される（すなわち、結果HTMLコードをレンダリングするために追加的な処理を行わない）標準HTML開始タグである。HTML構文において、HTMLファイルの始まりを示し、これもまたリテラルであるライン２１の終了タグ</html>と対になっている。

【００４０】コード宣言ブロックは、ファイル４００のライン３～１０に存在する。一般に、サーバ側コード宣言ブロックは、ページオブジェクトおよび制御オブジェクトメンバ変数ならびにサーバ上で実行処理されるメソッドを定義する。以下のフォーマットにおいて：

```
<script runat = "server" [language = "language"][src = "externalfile"]>
.....................................
</script>
```

ここで、言語およびsrcパラメータは任意である。本発明のある実施形態において、コード宣言ブロックは、"サーバ"に設定された値を有する"runat"属性を含む<script>タグを用いて定義される。任意には、"language"属性を内コードのシンタックスを特定するために用いてもよい。デフォルト言語は、ページ全体の言語構成を表すことができるが、コード宣言ブロックの"language"属性により、開発者は、例えば、Jscript およびPERL (Practical Extraction and Report Language)などの同じウェブページ実行内で異なる言語を用いることができる。<script>タグはまた、任意には"src"ファイルを特定することができ、"src"ファイルは、そこからページコンパイラによる処理のための動的コンテンツ資源にコードが挿入される外部のファイルである。開示されている文法が本実施形態において用いられているが、別の実施形

態では、本発明の範囲内で異なる文法を用いることができることを理解すべきである。

【００４１】図４において、２つのサブルーチン、AddButton#ClickおよびDeleteButton#Clickがコード宣言ブロック内においてVisual Basicフォーマットで宣言されている。いずれのサブルーチンも２つの入力パラメータ、"Source"および"E"をとり、クライアント側クリックイベントが対応のボタンに検知されるとHTTPリクエストへのレスポンスでサーバ上で実行処理される。AddButton#Clickサブルーチンにおいて、ユーザ名テキストボックスでのテキストは、単語"Add"に連結され、メッセージのテキストデータメンバにロードされる。DeleteButton#Clickサブルーチンにおいて、ユーザ名テキストボックスでのテキストは、単語"Delete"に連結され、メッセージのテキストデータメンバにロードされる。図４に示していないが、サーバ側制御オブジェクトのメンバ変数は、ファイル４００のコード宣言ブロックで宣言され得る。例えば、Visual Basicシンタックスを用いると、キーワードスペース"DIM"は、サーバ側制御オブジェクトのデータ変数を宣言する。

【００４２】「コードレンダリングブロック」（図示せず）もまた、動的コンテンツ資源に含まれ得る。本発明のある実施形態において、コードレンダリングブロックは、ページレンダリング時に実行処理される１つの「レンダリング」メソッドで実行処理する。コードレンダリングブロックは、以下のフォーマット（他のフォーマットも別の実施形態において考えられるが）を満たす。

【００４３】<% InlineCode %>

ここで“InlineCode”は、レンダリング時にサーバ上で実行処理する独立言語型コードブロックまたは制御フローブロックを含んでいる。

【００４４】インライン表現もまた、以下のような例示的なシンタックスを用いてコードレンダリングブロック内で用いることができる。

【００４５】<%= InlineExpression %>

ここで、“InlineExpression”ブロックに含まれる表現は、“InlineExpression”からの値を宣言内の適切な場所のホールダに書込むページオブジェクトの"Response.Write(InlineExpression)"への呼び出しによって最終的に包含される。例えば、“InlineExpression”は、以下のようなコードレンダリングブロックに含まれ得る。

【００４６】<font size = "<%=x%>" > Hi <%=Name%>, you are <%=Age%>!

これは、値"x"に格納されるフォントで挨拶およびある人の年齢についての記述を出力する。その人の名前および年齢は、コード宣言ブロック（図示せず）においてストリングとして定義される。結果HTMLコードは、適切な場所に“InlineExpression”の値を含むようにサーバでレンダリングされHTTPレスポンスでクライアントへ送信される。すなわち、以下のようなものである。

【００４７】<font size = "12" > Hi Bob, you are 35 !

ファイル４００のライン１１において、<body>は、HTMLドキュメントの本文の始まりを規定するための標準HTMLタグである。フィイル４００のライン２０において、終了タグ</body>もまた示されている。本発明のある実施形態において、<body>および</body>のいずれもリテラルである。

【００４８】HTMLフォームブロックの開始タグ<form>が、図４のファイル４００の本文セクション内、ライン１２に見られる。フォームブロックの終了タグ</form>は、HTMLファイル４００のライン１９に見られる。任意のパラメータ"id"もまた、所与の識別子をフォームブロックに関連付けるためにHTML制御タグに含むことができ、これによって、１つのHTMLファイルに多数のフォームブロックが含まれることが可能になる。

【００４９】ファイル４００のライン１８において、「メッセージ」によって識別されたサーバ側ラベルが宣言される。「メッセージ」ラベルは、ウェブページ上にラベルを表示するために、ファイル４００のライン５および８で宣言されたコードで用いられる。

【００５０】フォームブロック内に、図１のユーザインタフェース要素１０６、１０８および１１０に対応する３つのHTML制御タグ例が示されている。第１のユーザインタフェース要素がテキストボックスに相当するファイル４００のライン１３で宣言される。テキストリテラル"User Name"は、テキストボックスの左側に位置するラベルを宣言する。type="Text"を有する入力タグは、テキストボックスクライアント側ユーザインタフェース要素をレンダリングするサーバ側制御オブジェクトとして"UserName"という識別子を有するテキストボックスサーバ側制御オブジェクトを宣言する。ファイル４００のライン１５および１６は、それぞれ、図１のボタン１０８および１１０として示されるクライアント側ユーザインタフェース要素を宣言する。"OnServerClick"パラメータは、ファイル４００のコード宣言ブロックで宣言された適切なサブルーチンを特定する。そして、ファイル４００での宣言へのレスポンスで生成されたサーバ側ボタン制御オブジェクトが、クライアント側ボタンのHTMLコードおよびボタンクリックイベントを実行する関連のサーバ側コードをレンダリングする。

【００５１】ファイル４００で宣言されたテキストボックスおよびボタンは、HTMLサーバ制御宣言の例である。初期状態では、ASP+リソース内のすべてのHTMLタグはリテラルテキストコンテンツとして取り扱われ、ページ開発者のプログラミングにおいてアクセスすることができない。しかし、ページ開発者は、"server"に設定された値を有する"runat"属性を用いて指定することによって、HTMLタグは構文解析され、アクセス可能サーバ制御宣言として扱われるべきであることを示すことができ

る。任意には、各サーバ側制御オブジェクトは、対応する制御オブジェクトのプログラム参照を可能にする一意の"id"属性と関連付けることができる。サーバ側制御オブジェクト上のプロパティ引数およびイベント結合もまた、タグ要素における属性対である宣言名／値を用いて特定することができる（例えば、OnServerClick は"MyButton#Click"対に等しい）。

【００５２】本発明のある実施形態において、HTML制御オブジェクトを宣言する一般的なシンタックスは以下のとおりである。

【００５３】

```
<HTMLTag id = "Optional Name" runat = server>
.....
</HTMLTag>
```

ここで、"Optional Name"は、サーバ側制御オブジェクトの一意の識別子である。現在サポートされているHTMLタグのリストならびに関連シンタックスおよびCOM+クラスを表１に示すが、他のHTMLタグも本発明の範囲内で考えることができる。

【００５４】

【表１】

HTML タグ名	例	COM+ クラス
<a>	 My Link 	AnchorButton
<img>	<img id = "MyImage" runat = server>	Image
		Label
	Some contents	Panel
<form>	<form id = "MyForm" runat = server> </form>	FormControl
<select>	<select id = "MyList" runat = server> <option>One</option> <option>Two</option> <option>Three</option> </select>	DropDownList
<input type = file>	<input id = "MyFile" type = file runat = server>	FileInput
<input type = text>	<input id = "MyTextBox" type = text>	TextBox
<input type = password>	<input id = "MyPassword" type = password>	TextBox
<input type = reset>	<input id = "MyReset" type = reset>	Button
<input type = radio>	<input id = "MyRadioButton" type = radio runat = server>	RadioButton
<input type = checkbox>	<input id = "MyCheck" type = checkbox runat = server>	CheckBox
<input type = hidden>	<input id ="MyHidden" type = hidden runat = server>	HiddenField
<input type = image>	<input type = image src = "foo.jpg" runat = server>	ImageButton
<input type = submit>	<input type = submit runat = server>	Button
<input type = button>	<input type = button runat = server>	Button
<button>	<button id = MyButton runat = server>	Button
<textarea>	<textarea id = "MyText" runat = server> This is some sample text </textarea>	TextArea

【００５５】標準HTML制御タグに加えて、本発明のある実施形態によって、開発者は、HTMLタグセットの外側に共通のプログラム機能についてカプセル化する再利用可能コンポーネントを作成することが可能になる。これらのカスタムサーバ側制御オブジェクトは、ページファイル内の宣言タグを用いて特定される。カスタムサーバ側制御オブジェクト宣言は、"server"に設定された値を有する"runat"属性を含む。任意には、カスタム制御オブジェクトのプログラム参照を可能するために、一意の"id"属性が特定される。さらに、タブ要素の宣言名／値属性対は、サーバ側制御オブジェクトのプロパティ引数およびイベント結合を特定する。インラインテンプレートパラメータもまた、適切な「テンプレート」接頭辞子要素を親サーバ制御オブジェクトに与えることによってサーバ側制御オブジェクトに結合されることがある。カスタムサーバ側制御オブジェクト宣言のフォーマットは、次のようになる。

【００５６】<servercntrlclassname id="OptionalName"〔propertyname="propval"〕runat=server/>

ここで、"servercntrlclassname"は、アクセス可能制御クラスであり、"OptionalName"は、サーバ側制御オブジェクトの一意の識別子であり、"propval"は、制御オブジェクトの任意のプロパティ値を表す。

【００５７】別の宣言文法を用いて、XMLタグ接頭語

は、以下のフォーマットを用いることにより、ページ内にサーバ側制御オブジェクトを特定するためのより簡潔な表記法を提供するのに用いることができる。

【００５８】<tagprefix:classname id = "OptionalName" runat = server/>

ここで、“tagprefix”は、所与の制御名スペースライブラリと関連し、“classname”は、関連名スペースライブラリでの制御の名前を表す。任意の“propertyvalue”もまたサポートされている。

【００５９】要するに、本発明の実施形態は、クライアントに送るHTMLコードを生成するためにサーバ上で作成および実行処理されるサーバ側制御オブジェクトを含む。HTMLコードは、いかなる有効なHTML構成も具現化することができ、例えば、ACTIVEXタイプの制御、JAVA アプレット、スクリプト、その他クライアントでのユーザインタフェースボタンおよび他のユーザインタフェース要素を生成するいかなるウェブ資源も参照することができる。クライアントでのユーザは、サーバ側制御オブジェクトに論理的に対応するこれらのユーザインタフェース要素と対話し、リクエストをサーバに送り返すことができる。サーバ側制御オブジェクトは、サーバ上で再作成され、クライアントにレスポンスとして送信すべき次のラウンドのHTMLコードを生成するように、ユーザインタフェース要素のデータ、イベントおよび他の特性を処理する。

【００６０】図５を参照すると、本発明の実施形態のコンピュータシステムの一例は、プロセッサユニット５０２、システムメモリ５０４およびシステムメモリ５０４を含む種々のシステムコンポーネントをプロセッサユニット５００に接続するシステムバス５０６を含んでいる従来のコンピュータシステム５００という形態の汎用コンピュータ装置を含んでいる。システムバス５０６は、メモリバスまたはメモリコントローラ、種々のバスアーキテクチャを用いるペリフェラルバスおよびローカルバスを含む幾つかのタイプのバス構造のいずれであってもよい。システムメモリは、再生専用メモリ（ROM）５０８およびランダムアクセスメモリ（RAM）５１０を含んでいる。コンピュータシステム５００内の要素間での情報の転送を助ける基本ルーチンを含んでいる基本入力／出力システム５１２（BIOS）は、ROM５０８に格納されている。

【００６１】コンピュータシステム５００は、さらに、ハードディスクの読み出しおよび書き込みを行うハードディスクドライブ５１２、着脱可能な磁気ディスク５１６の読出しおよび書込みを行う磁気ディスクドライブ５１４およびCD ROM、DVDまたは他の光学媒体のような着脱可能な光ディスク５１９の読出しおよび書込みを行う光ディスクドライブ５１８を含んでいる。ハードディスクドライブ５１２、磁気ディスクドライブ５１４および光ディスクドライブ５１８は、それぞれ、ハードディスクドライブインタフェース５２０、磁気ディスクドライブインタフェース５２２および光ディスクドライブインタフェース５２４によってシステムバス５０６接続されている。ドライブおよびその関連コンピュータ読み取り可能媒体が、コンピュータシステム５００のコンピュータ読み取り可能命令、データ構造、プログラムおよび他のデータの不揮発性記憶部を提供している。

【００６２】本明細書に記載の上記環境例では、ハードディスク、着脱可能な磁気ディスク５１６および着脱可能な光ディスク５１９を用いているが、データ保存可能な他のタイプのコンピュータ読み取り可能媒体を上記システム例に用いることができる。上記動作環境例に用いることができるこれらの他のタイプのコンピュータ読み取り可能媒体は、例えば、磁気カセット、フラッシュメモリカード、デジタルビデオディスク、ベルヌイ(Bernoulli)カートリッジ、ランダムアクセスメモリ（RAM）および再生専用メモリ（ROM）などがある。

【００６３】多数のプログラムモジュールが、ハードディスク、磁気ディスク５１６、光ディスク５１９、ROM５０８またはRAM５１０に格納され、これらは、オペレーティングシステム５２６、１つ以上のアプリケーションプログラム５２８、他のプログラムモジュール５３０およびプログラムデータ５３２を含む。ユーザは、コマンドおよび情報をコンピュータシステム５００にキーボード５３４およびマウス５３６または他のポインティング装置などの入力装置によって入力することができる。他の入力装置としては、例えば、マイクロフォン、ジョイステック、ゲームパッド、サテライトディッシュおよびスキャナなどがある。これらおよび他の入力装置は、システムバス５０６に接続されているシリアルポートインタフェース５４０を介して処理装置５０２に接続されていることが多い。しかし、これらの入力装置はまた、パラレルポート、ゲームポートまたはユニバーサルシリアルバス（USB）などの他のインタフェースによって接続されていてもよい。モニタ５４２または他のタイプの表示装置もまた、ビデオアダプタ５４４などのインタフェースを介してシステムバス５０６と接続している。モニタ５４２に加えて、コンピュータシステムは、典型的には、スピーカおよびプリンタなどの他の周辺出力装置（図示せず）を含む。

【００６４】コンピュータシステム５００は、リモートコンピュータ５４６のような１つ以上のリモートコンピュータへの論理接続を用いたネットワーク化された環境で動作することができる。リモートコンピュータ５４６は、コンピュータシステム、サーバ、ルータ、ネットワークPC、ピア（peer）装置、または他の共通ネットワークノードであり得、典型的にコンピュータシステム５００との関連で上述した要素の多くまたはすべてを含む。ネットワーク接続は、ローカルエリアネットワーク（LAN）５４８およびワイドエリアネットワーク（WAN）５５

０を含む。このようなネットワーク環境は、オフィス、企業規模コンピュータネットワーク、イントラネットおよびインターネットにおいて珍しいものではない。

【００６５】LANネットワーク環境で用いるとき、コンピュータシステム５００は、ネットワークインタフェースまたはアダプタ５５２を介してローカルネットワーク５４８に接続される。WANネットワーク環境で用いるとき、コンピュータシステム５００は、典型的に、インターネットのようなワイドエリアネットワーク５５０による通信を確立するためのモデム５５４または他の手段を含む。モデム５５４は内蔵または外付けのいずれでもよく、シリアルポートインタフェース５４０を介してシステムバス５０６と接続されている。ネットワーク化された環境において、コンピュータシステム５００に関連して述べたプログラムモジュールまたはその一部は、リモートメモリ記憶装置に記憶されてもよい。図示されたネットワーク接続は例であって、コンピュータ間の通信リンク確立のために他の手段を用いることができる。

【００６６】本発明の実施形態において、コンピュータ５００は、ウェブサーバを表し、CPU５０２が、記憶媒体５１６、５１２、５１４、５１８、５１９またはメモリ５０４のうち少なくとも１つに記憶されたASP+リソース上でページファクトリモジュールを実行処理する。HTTPレスポンスおよびリクエストは、クライアントコンピュータ５４６に接続されたLAN５４８により通信される。

【００６７】図６は、本発明のある実施形態におけるページオブジェクトおよび他の制御オブジェクトのサーバ側処理を表すプロセスフローチャートである。処理６００において、ページオブジェクト構築部が、ページファクトリモジュール３０８によって呼び出される（図３参照）。その結果、ページオブジェクト（例えば、図３におけるページオブジェクト３１４を参照）は、クライアント上のウェブページユーザインタフェース要素と論理的に対応するように作成される。処理６０２において、ページファクトリモジュールは、クライアントから受け取ったHTTPリクエストの段階的な処理を始めるページオブジェクトのProcessRequest（）メンバ関数を呼び出す。本発明の一実施形態の第１の段階において、サーバ側作成処理(図示せず)は、ページオブジェクトの制御オブジェクト階層に含まれる子孫サーバ側制御オブジェクトの作成である。すなわち、子制御オブジェクトの構築部がHTTPリクエスト処理の処理生存期間の間制御オブジェクトを作成するために繰り返し呼び出される。

【００６８】しかし、別の実施形態において、子制御オブジェクトの作成は、制御オブジェクトが所与の処理ステップ（例えば、ポストバックイベントのハンドリング、ポストバックデータのハンドリング、ビューステートのローディングおよび保存、データ結合の分解または対応するユーザインタフェース要素のHTMLコードのレンダリング）に必要とされるまで遅らせることができる。「制御オブジェクト作成を遅らせる」後者の実施形態は、不必要なCPUおよびメモリの利用を減らすことができるので最適である。例えば、クライアントから受け取ったユーザ入力イベントが、全く異なるウェブページの作成ということになる場合がある。この場合、直ちに制御オブジェクト階層の終了させ、新たなページの新たな異なる制御オブジェクト階層を例示することになるイベントを処理するためだけに、以前のページの制御オブジェクト階層全体を例示する必要はない。

【００６９】ページオブジェクトのProcessRequestメソッドのサーバ呼び出しに応答して、処理６０４～６２０は、所与のHTTPリクエストのデータなどに応じ、データページオブジェクトおよび個々の子孫制御オブジェクトによって実行処理することができる。本発明のある実施形態において処理６０４～６２０を図６の順序で各個々のオブジェクトに対し行う。しかし、１つのオブジェクトに対する所与の処理は、HTTPリクエストによっては、別のオブジェクトの所与の処理に関して順番に行われなかったり、全く処理が行われないこともある。例えば、第１のオブジェクトは、初期化処理６０４およびそのロード処理６０６を行い、ポストバックデータ処理６０８を開始し、その後、遅れた制御オブジェクト作成により子孫制御オブジェクトが、それ自体の初期化処理６０４およびロード処理６０６を行う。ページオブジェクトおよび子孫制御オブジェクトによる処理の順番は、これには限らないが、HTTPリクエストにおけるデータの性質、制御オブジェクト階層の構成、制御オブジェクトの現在の状態および制御オブジェクト作成が遅れて行われるかどうかを含む種々の要因による。

【００７０】初期化処理６０４は、動的コンテンツ資源における初期化に関する任意のサーバ側コードを実行処理することによって制御オブジェクトが作成された後、制御オブジェクトを初期化する。このようにして、各サーバ側制御オブジェクトは、動的コンテンツ資源で宣言された特定のサーバ側機能でカスタマイズされ得る。本発明の実施形態において、動的コンテンツコードは、サーバ上のASP+リソースでページ開発者によって宣言されたベースページ制御クラスをカスタマイズまたは拡張するものである。ASP+リソースがコンパイルされると、宣言されたコードは、適切な初期コード（例えば、ページオブジェクトおよび子孫制御オブジェクトの初期化（）メソッド）に含まれている。初期化処理６０４は、このコードを実行処理して、ページベースクラスおよび子孫制御オブジェクトのベースクラスをカスタマイズまたは拡張する。

【００７１】本発明のある実施形態において、サーバ側制御オブジェクトの状態管理は、サーバ側制御オブジェクトを以前の状態に戻すことによってクライアント・サーバシステムの無状態モデルを収納するために搬送可能

状態構造を用いるロード処理６０６および保存処理６１６においてサポートされる。ある実施形態では、状態は、一対のHTTPリクエスト／レスポンスの隠れた１つ以上のHTMLフィールドでサーバを往復するが、他の搬送可能状態構造も本発明の範囲内であると考えられる。

【００７２】クライアントとサーバ間の現在のページに関する所与のリクエストおよびレスポンスによる一連の処理において、１つ以上の制御オブジェクトの状態は、先のリクエストの処理後に保存処理６１６によって搬送可能状態構造に記録される。本発明のある実施形態において、階層情報またはサーバが適切な制御オブジェクトと所与の状態とを関連付けることを可能にする制御オブジェクト識別子を含む追加の状態情報もまた、搬送可能状態構造に含まれる。次のHTTPリクエストにおいて、状態情報は、搬送可能状態構造でサーバに戻される。サーバは、受信した搬送可能状態構造から状態情報を抽出し、状態データを制御オブジェクト階層内の適切な制御オブジェクトにロードし、各制御オブジェクトを先のHTTPレスポンス以前に存在したような状態に戻す。現在のリクエストに対する処理後、１つ以上のサーバ側制御オブジェクトの状態は、再度保存処理６１６によって搬送可能状態構造に記録され、次のHTTPレスポンスで搬送可能状態構造をクライアントに戻す。

【００７３】ロード処理６０６の結果、各サーバ側制御オブジェクトを先のHTTPリクエスト以前の状態と同じ状態にする。例えば、テキストボックス制御オブジェクトが、先のHTTPレスポンス以前に"JDoe"と同等のプロパティ値を含む場合、テキストストリング"JDoe"をそのプロパティ値にロードすることなどによって、ロード処理６０６は、同じ制御オブジェクトを先の状態に戻す。さらに、所与のオブジェクトの状態が記憶され回復されたかどうかについても構成可能である。

【００７４】本発明の一実施形態を要約すると、１つ以上のサーバ側制御オブジェクトの状態が処理後「保存」される。保存された状態情報は、レスポンスによりクライアントに送信される。クライアントは、次のレスポンスで保存された状態情報をサーバ側に戻す。サーバは、階層の状態が以前の状態に戻るように、新たに例示されたサーバ側制御オブジェクト階層に状態情報をロードする。

【００７５】別の実施形態では、サーバ上、またはサーバからクライアント、そしてサーバへ戻るという往復の間、サーバによってアクセス可能な幾つかの他のウェブロケーションに状態情報を保持できる。クライアントリクエストをサーバが受信した後、この状態情報は、サーバによって取り出され、制御オブジェクト階層内の適切なサーバ側制御オブジェクトにロードされ得る。

【００７６】処理６０８において、HTTPリクエストから受け取ったポストバックデータが処理される。ポストバックデータは、対になったキー値、階層表示（例えば、XML）またはRDF（Resource Description Framework）のような他のデータ表示でのHTTPリクエストのペイロードに含まれ得る。処理６０８は、ペイロードを構文解析してサーバ側制御オブジェクトの一意の識別子を識別する。識別子（例えば、"page1:text1"）を見つけ、識別されたサーバ側制御オブジェクトが制御オブジェクト階層に存在する場合、対応するポストバックデータがその制御オブジェクトへと渡される。例えば、図１を参照すると、テキストボックス１０６およびテキスト"JDoe"に関連する一意の識別子は、HTTPリクエスト１１４のペイロードでウェブサーバ１１６へ送られる。処理６０８はHTTPリクエスト１１４のペイロードを構文解析し、テキストボックス１０６の一意の識別子とその関連値（すなわち"JDoe"）を得る。それから処理６０８は、テキストボックス１０６の一意の識別子を分解し、対応するサーバ側制御オブジェクトを識別し、処理するオブジェクトに"JDoe"値を渡す。

【００７７】ロード処理６０６に関して説明したように、サーバ側制御オブジェクトのプロパティ値は、以前の状態に戻され得る。ポストバックデータを受け取ると、サーバ側制御オブジェクトは、渡されたポストバック値が対応する先のプロパティ値を変化させたかどうかを判断する。変化した場合には、関連制御オブジェクトのデータ変化を示す変化リストにその変化をロギングする。すべてのポストバックデータが制御オブジェクト階層内で処理された後、制御オブジェクトメソッドが呼び出され、サーバ上で起動している株価検索アプリケーションのような１つ以上の非ユーザインタフェースサーバコンポーネントに対し、１つ以上のポストデータ変化イベントを提起し得る。ポストデータ変化イベントは、例えば、ポストバックデータがサーバ側制御オブジェクトのプロパティを変化させたということを示しているイベントである。例示的な実施形態において、このようなイベントは、システム提供イベント待ち行列に送られ、イベントを処理するよう登録されたサーバ側コードを呼び出すことができる。そして、サーバ側コードは、非ユーザインタフェースサーバコンポーネントのメソッドを呼び出し得る。このように、サーバ側非ユーザインタフェースサーバコンポーネントは、サーバ側制御オブジェクトのデータの変化によってトリガされたイベントに応答することができる。アプリケーション提供イベント待ち行列、ポーリング、および処理割込みを用いることを含む、イベントを実行する別の方法もまた本発明の範囲内で考えることができる。

【００７８】処理６１０において、ポストバックイベントをハンドリングする。ポストバックイベントは、HTTPリクエストのペイロードで通信され得る。処理６１０は、そのイベントが向けられているサーバ側制御オブジェクトを識別する特定のイベントターゲット（例えば、本発明のある実施形態では、"##EVENTTARGET"とラベル

付けされている）を構文解析する。さらに、処理６１０は、探し当てたイベント引数がある場合はそれを構文解析し、イベント引数（例えば、本発明のある実施形態では、"##EVENTARGUMENT"とラベル付けされている）を特定されたサーバ側制御オブジェクトに与える。制御オブジェクトは、動的コンテンツ資源に関連する非ユーザインタフェースサーバコンポーネント（例えば、サーバ側株価検索アプリケーション）のメソッドを呼び出すサーバ側コードによって処理するイベントを立ち上げる。

【００７９】処理６１２は、サーバ側制御オブジェクトとサーバがアクセス可能な１つ以上のデータベースとの間のデータ結合関係を解明し、これにより、データベース値で制御オブジェクトプロパティにおいて更新し、かつ／または制御オブジェクトプロパティの値でデータベースフィールドを更新する。本発明のある実施形態では、サーバ側制御オブジェクトのプロパティは、サーバ側アプリケーションデータベースの表のような親データ結合コンテナのプロパティと関連付けられ（またはデータ結合され）得る。データ結合処理６１２の間に、ページフレームワークは、対応する親データ結合コンテナプロパティの値を有するデータ結合された制御オブジェクトプロパティを更新することができる。このように、次のレスポンスにおけるウェブページ上のユーザインタフェース要素は、更新されたプロパティ値を正確に反映する。なぜなら、ユーザインタフェース要素が対応する制御オブジェクトプロパティは、データ結合処理６１２の間に、自動的に更新されたからである。同様に、制御オブジェクトプロパティはまた、親データ結合コンテナフィールドに更新され得、これにより、サーバ側制御オブジェクトからのポストバック入力でサーバ側アプリケーションデータベースを更新する。

【００８０】処理６１４は、制御オブジェクト状態が保存され出力がレンダリングされる以前に実行処理され得る多数の更新処理を行う。処理６１６は、制御オブジェクト階層内の１つ以上の制御オブジェクトから状態情報（すなわち、ビューステート）を要求し、状態情報を格納し、HTTPレスポンスペイロードでクライアントへ送られる搬送可能状態構造に挿入する。例えば、“grid”制御オブジェクトは、値のリストの現在のインデックスページを保存し、“grid”制御オブジェクトは、次のHTTPリクエスト（すなわち、処理６０６）の後この状態に戻ることができる。上記のように、ビューステート情報は、クライアントによる次のアクション以前の制御オブジェクト階層の状態を表す。ビューステート情報が戻ってくると、それは、制御オブジェクト階層を、クライアントポストバック入力処理またはデータ結合以前の先の状態にするのに用いられる。

【００８１】レンダリング処理６１８は、HTTPレスポンスでクライアントへ送る適切なオーサリング言語出力（例えば、HTMLデータ）を生成する。レンダリングは、すべてのサーバ側制御オブジェクトのトップダウン階層ツリーウォークおよび埋め込まれたレンダリングコードにより行われる。処理６２０は、任意の最終のクリーンアップ作業（例えば、ファイルを閉じたりデータベースの接続）を行い、制御オブジェクト階層を終了させる。次いで、処理は６０２に戻り、処理６２２を行い、そこでページオブジェクトは、その破壊部を呼び出すことによって終了する。

【００８２】図７は、本発明のある実施形態におけるサーバ側制御クラスの一例の表記を表している。サーバ側制御クラスは、本発明のある実施形態におけるすべてのサーバ側制御オブジェクトに共通のメソッド、プロパティおよびイベントを定義している。より具体的な制御クラス（例えば、ウェブページでのクライアント側ボタンに対応するサーバ側ボタン制御オブジェクト）は、この制御クラスから派生している。示された制御クラス７００は、プロパティ７０２およびメソッド７０４を格納するメモリを含む。データメンバとメソッドとの組み合わせが異なる他の制御クラスの実施形態もまた、本発明の範囲内で考えられる。

【００８３】示された実施形態において、プロパティ７０２はパブリックである。プロパティ"ID"は、制御オブジェクト識別子を示している読み取りおよび書き込み可能なストリング値である。プロパティ"Visible"は、対応しているクライアント側ユーザインタフェース要素のオーサリング言語データをレンダリングすべきかどうかを示す読み取りおよび書き込み可能なブーリアン値である。プロパティ"MaintainState"は、制御オブジェクトが、現在のページリクエストの終了時に（すなわち、図６の保存処理６１６へのレスポンスで）そのビューステート（およびその子オブジェクトのビューステート）を保存すべきかどうかを示している読み取りおよび書き込み可能なブーリアン値である。プロパティ"Parent"は、制御オブジェクト階層の現在の制御オブジェクトを関連する制御コンテナ（図８参照）への読み取り可能なリファレンスである。プロパティ"Page"は、現在の制御オブジェクトがホストされているルートページオブジェクトへの読み取り可能なリファレンスである。プロパティ"Form"は、現在の制御オブジェクトがホストされているフォーム制御オブジェクトへの読み取り可能なリファレンスである。プロパティ"Trace"は、開発者のトレースログの書き込みを可能にする読み取り可能なリファレンスである。プロパティ“ BindingContainer ”は、制御オブジェクトの直接データ結合コンテナへの読み取り可能なリファレンスである。プロパティ"Bindings"は、制御オブジェクトデータ結合連関の集合への読み取り可能なリファレンスである。

【００８４】メソッド７０４は、リクエストの処理および制御オブジェクトのデータメンバへのアクセス方法を含む。ある実施形態において、メソッドは、制御オブジ

ェクトのメモリスペースに格納されるポインタによって参照される。この参照手段ははまた、コンテナ制御オブジェクト、制御コレクションオブジェクト、およびページオブジェクトを含む他のサーバ側オブジェクトの実施形態において用いられる。メソッド"Init()"は、子制御オブジェクトが作成された後、これらを初期化するのに用いられる（図６の処理６０４を参照）。メソッド"Load()"は、先のHTTPリクエストからビューステート情報を回復するのに用いられる（図６の処理６０６を参照）。メソッド"Save()"は、後のHTTPリクエストで用いるビューステート情報を保存するのに用いられる（図６の処理６１６を参照）。メソッド"PreRender()"は、ビューステートの保存およびコンテンツのレンダリングに先立って必要なプレレンダリングステップを行うのに用いられる（図６の処理６１４を参照）。メソッド"Render(TextWriter output)"は、現在の制御オブジェクトに対応するユーザインタフェース要素のオーサリング言語コードを出力するのに用いられる（図６の処理６１８を参照）。コードは、クライアントへのレスポンスでコードを格納するために出力ストリームを通してやり取りされる（「出力」パラメータでそれを渡す）。メソッド"Dispose()"は、制御オブジェクトを終了する前に最終的なクリーンアップ作業を行うのに用いられる（図６の処理６２０を参照）。

【００８５】メソッド"GetUniqueID()"は、制御オブジェクトの一意の、階層的に認定されたストリング識別子を得る。メソッド"GetControlWithID(String id)"は、与えられた識別子("id")を有する直接の子制御オブジェクトへのリファレンスを戻す。メソッド"GetControlUniqueWithID(String id)"は、一意の階層識別子("id")を有する子制御オブジェクトへのリファレンスを戻す。

【００８６】メソッド"PushControlPropertyTwoBindingContainer (String prop Name)"は、ポストバックデータ値がサーバ側制御オブジェクト内で変化するとポストバックデータからの２方向データ結合の結合コンテナを更新するのに用いられる。メソッド"PushBindingContainerPropertyTwoControl(String prop Name)"は、現在の結合コンテナ値をもつサーバ側制御オブジェクトプロパティを更新するのに用いられる。メソッド"HasBindings()"は、制御オブジェクトが結合連関を有しているかどうかを示しているブーリアン値を戻す。これらの３つの機能は、サーバ側制御オブジェクトのプロパティとサーバ側データ格納部における属性との結合関係を解明するのに用いられる（図６のデータ結合処理６１２を参照）。

【００８７】図８は、本発明のある実施形態におけるサーバ側コンテナ制御クラスの一例を示す。コンテナ制御クラスは、入れ子の子制御オブジェクトをサポートし、搬送可能状態構造へビューステート情報を自動的に直列化および非直列化する構成を提供する。コンテナ制御クラス８００は、プロパティ８０２およびメソッド８０４を格納するメモリを含んでいる。プロパティ"Controls"は、制御オブジェクト階層内の子制御オブジェクトの"ControlCollection"への読み取り可能なリファレンスである（図９を参照）。プロパティ"StateCollection"は、多数のページリクエストにわたって制御オブジェクトの状態を維持するのに用いられるビューステート情報の辞書への読み取り可能なリファレンスである。メソッド"HasControls()"は、制御オブジェクトが子制御オブジェクトを有しているかどうかを示しているブーリアン値を戻す。

【００８８】別の実施形態において、コンテナ制御クラス８００のメンバプロパティおよびメソッドは、各制御オブジェクトがそれ自体、子オブジェクトをサポートできるように図７の制御クラス７００に組み込まれてもよい。しかし、このような実施形態の制御オブジェクトが、このような子オブジェクトを含まないなら、（制御コレクションタイプの）制御メンバは空っぽであるだろう（すなわち、子オブジェクトを含まない）。

【００８９】図９は、本発明の実施形態におけるサーバ側制御コレクションクラスの一例を示す。制御コレクションクラス９００は、プロパティ９０２およびメソッド９０４を格納するメモリを含んでいる。プロパティ"All"は、インデックスによって命令された現在の制御オブジェクトのすべての子制御オブジェクトの読み取りおよび書き込み可能なスナップショットアレイである。プロパティ"this[index]"は、制御コレクション内の順序インデックスの制御オブジェクトへの読み取り可能なリファレンスである。プロパティ"Count"は、コレクション内の子制御オブジェクトの数を示している読み取り可能値である。

【００９０】メソッド"Add'Control child)"は、現在のコレクションに特定の制御オブジェクトを追加するのに用いられる。メソッド"IndexOf(Control child)"は、コレクション内の特定された子制御オブジェクトの順序インデックスを戻す。メソッド"GetEnumerator(boolAllowRemove)"は、コレクション内のすべての子制御オブジェクトのエニュメレータを戻す。メソッド"Remove(Control value)"は、現在のコレクションから特定の制御オブジェクトを取り除く。メソッド"Remove(int index)"は、与えられたインデックスに基づき現在のコレクションから特定の制御オブジェクトを取り除く。メソッド"Clear()"は、現在のコレクションからすべての制御オブジェクトを取り除く。

【００９１】図１０は、本発明のある実施形態におけるサーバ側ページオブジェクトの一例を示す。ページベースクラスは、本発明の実施形態内でのすべてのサーバ側操作ページに共通のメソッド、プロパティおよびイベントを定義する。ページオブジェクト１０００は、プロパティ１００２およびメソッド１００４を格納しているメ

モリを含んでいる。プロパティ"ErrorPage"は、未処理ページ例外のイベントでレンダリングされる読み取りおよび書き込み可能なストリングである。このように、ページオブジェクトは、HTTPレスポンスでクライアントに読み取り可能なエラーページを戻す。プロパティ"Requests"は、HTTPRequestへの読み取り可能なリファレンスであり、HTTPRequestは、入ってくるHTTPリクエストデータにアクセスするための機能性を与えるウェブサーバフレームワークによって与えられる。

【００９２】プロパティ"Response"は、クライアントへHTTPレスポンスデータを送信するための機能を提供するウェブサーバフレームワークによって与えられるHTTPレスポンスへの読み取り可能なリファレンスである。プロパティ"Application"は、ウェブサーバフレームワークによって与えられるHTTPApplicationへの読み取り可能なリファレンスである。プロパティ"Session"は、ウェブサーバフレームワークによって与えられるHTTPSessionへの読み取り可能なリファレンスである。プロパティ"Server"は、アプリケーションサーバページ互換性ユティリティオブジェクトであるサーバオブジェクトへの読み取り可能なリファレンスである。プロパティ"Context"は、ウェブサーバフレームワークによって与えられるウェブサーバオブジェクトのすべてへの読み取り可能なリファレンスであり、これにより、開発者は、さらなるパイプラインモジュール露出オブジェクトへのアクセスが可能になる。プロパティ"IsFirstLoad"は、ページオブジェクトが、初めて、またはクライアントポストバックリクエストへのレスポンスでロードおよびアクセスされているかどうかを示す読み取り可能なブーリアン値である。

【００９３】プロパティ"Load()"は、ページオブジェクトを初期化し、先のページリクエストからビューステート情報を回復するのに用いられる。プロパティ"Save()"は、後のページリクエストに用いるビューステート情報を保存するのに用いられる。プロパティ"HandleError（Exception errorInfo）"は、ページ実行処理の間に起こる未処理エラーを扱うのに用いられる。このイベントにおいて、ベースクラス実行は、クライアントをデフォルトエラーウェブページを有するURLへ向けなおす。メソッド"GetPostbackScript(Control target, String name, String arg)"は、所与の制御オブジェクトと関連するクライアント側スクリプトメソッドを戻す。メソッド"RegisterClientScriptBlock(String key, String script)"は、クライアントへ送信されるクライアント側スクリプトコードの重複ブロックを排除するのに用いられる。重複ブロックは同じキー値を有するスクリプトである。メソッド"IHTTPHandler.ProcessRequest(HTTPContext Context)"は、ウェブリクエストを処理するのに用いられる。IHTTPHandler.ProcessRequestは、クライアントから受信されたHTTPリクエストの処理を初期化するために呼び出される（図６の処理６０２を参照）。メソッド"IHTTPHandler.IsReusable()"は、ページオブジェクトが、多数のウェブリクエストを処理するのに再利用できるかどうかを示している。

【００９４】本明細書に記載の本発明の実施形態は、１つ以上のコンピュータシステムの論理ステップとして実行される。本発明の論理処理は、（１）１つ以上のコンピュータシステムで実行処理されるプロセッサ実行ステップシーケンスとして、（２）１つ以上のコンピュータシステム内の相互接続された機械モジュールとして実行される。実行は選択の問題であり、本発明を実行するコンピュータシステムの性能要件に依存する。したがって、本明細書に記載の本発明の実施形態を構成する論理処理は、処理、ステップ、オブジェクトまたはモジュールと様々に表現することができる。

【００９５】上記の明細書、実施例およびデータは、本発明の実施形態の構造および使用の完全な説明を提供している。本発明は、本発明の精神および範囲から逸脱することなく多数の形態で実施することができるので、本発明は添付の請求項にある。

【００９６】

【発明の効果】

【図面の簡単な説明】

【図１】　本発明のある実施形態におけるクライアントに表示するウェブページコンテンツを動的に生成するウェブサーバを示す。

【図２】　本発明のある実施形態におけるサーバ側制御オブジェクトを用いてクライアント側ユーザインタフェース要素の処理およびレンダリングのための処理のフローチャートを示す。

【図３】　本発明のある実施形態で用いるウェブサーバのモジュールの一例を示す。

【図４】　本発明のある実施形態における動的コンテンツリソース（例えば、ASP+リソース）の一例を示す。

【図５】　本発明のある実施形態の実行に有用なシステムの一例を示す。

【図６】　本発明のある実施形態におけるページオブジェクトの処理を表すフローチャートを示す。

【図７】　本発明のある実施形態におけるサーバ側制御クラスの一例を示す。

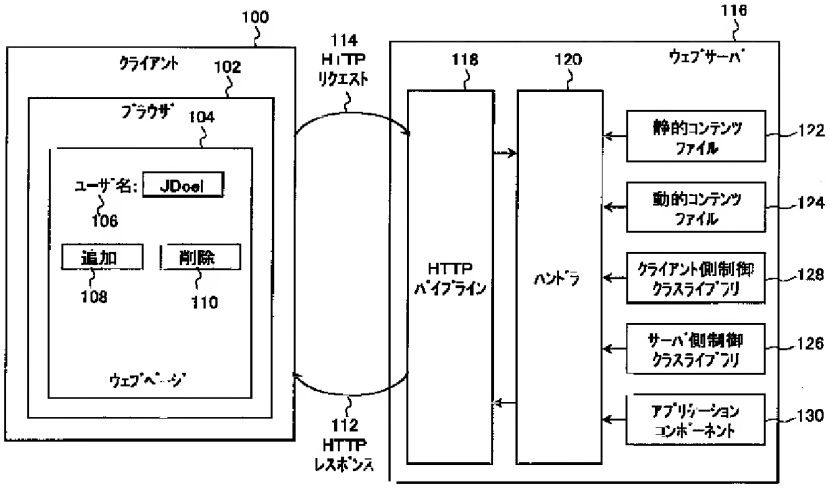
【図８】　本発明のある実施形態におけるサーバ側コンテナ制御クラスの一例を示す。

【図９】　本発明のある実施形態におけるサーバ側制御コレクションクラスの一例を示す。

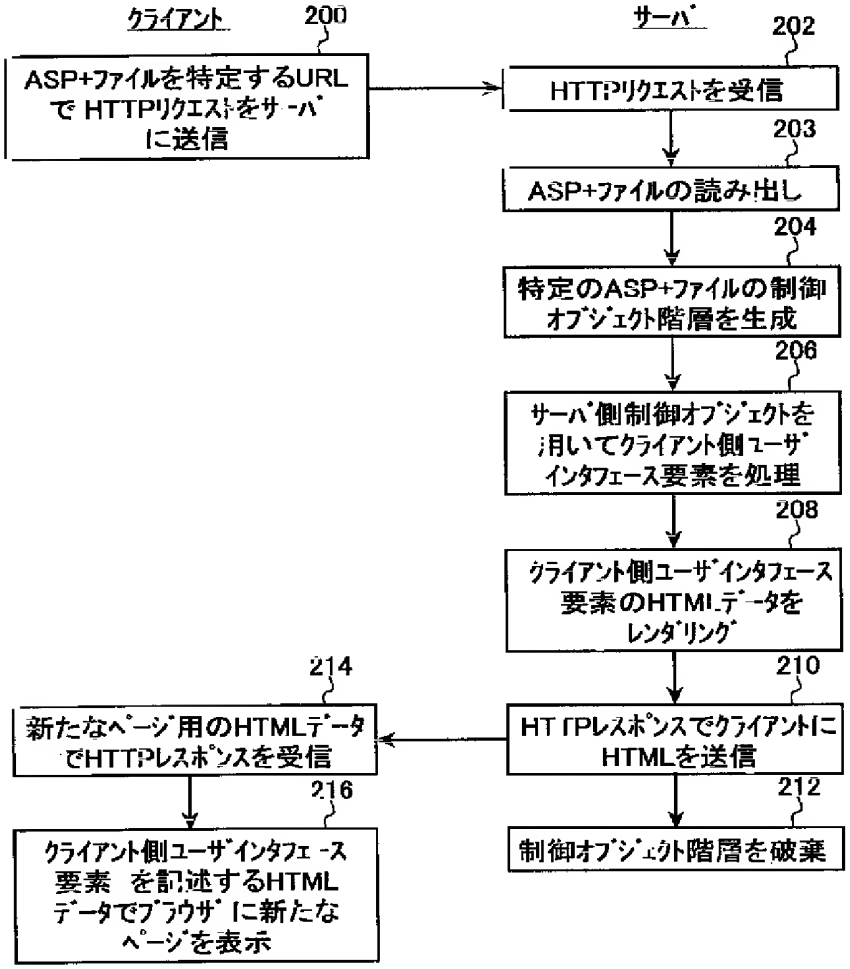
【図１０】　本発明のある実施形態におけるサーバ側ページクラスの一例を示す。

【符号の説明】

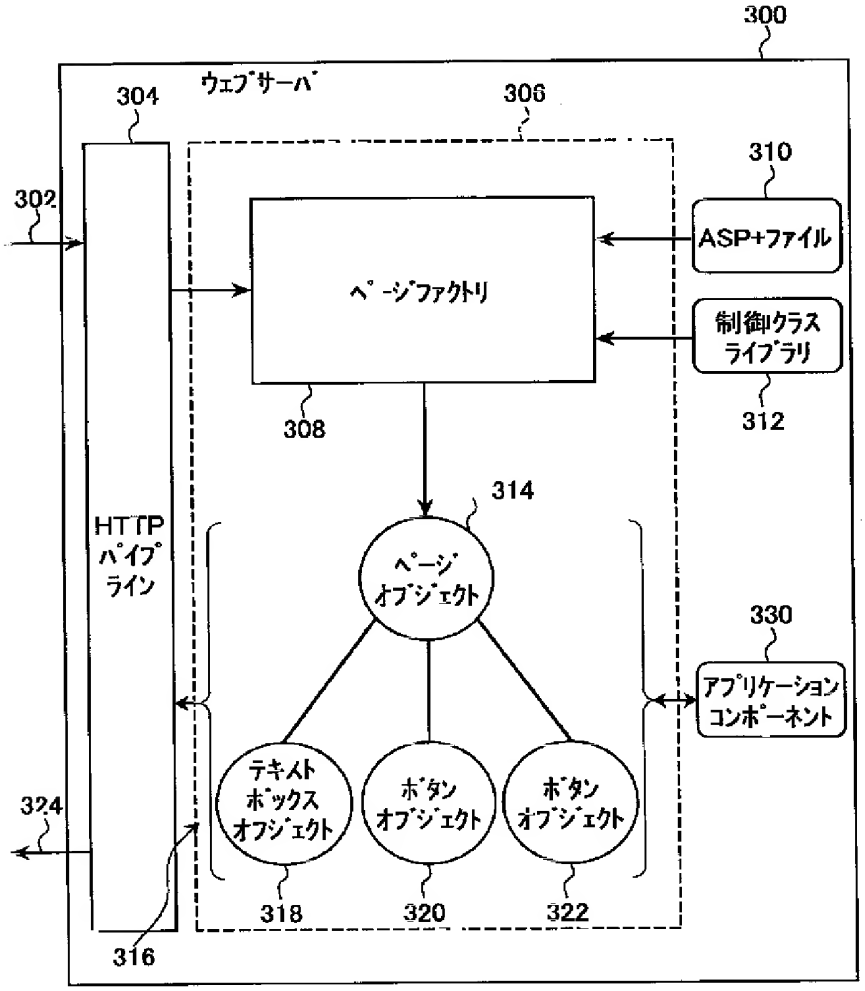
【図１】


100
クライアント
102
ブラウザ
104
ユーザ名:
JDoel
106
追加
108
削除
110
ウェブページ
114
HTTP
リクエスト
112
HTTP
レスポンス
116
ウェブサーバ
118
HTTP
パイプライン
120
ハンドラ
静的コンテンツ
ファイル
122
動的コンテンツ
ファイル
124
クライアント側制御
クラスライブラリ
128
サーバ側制御
クラスライブラリ
126
アプリケーション
コンポーネント
130

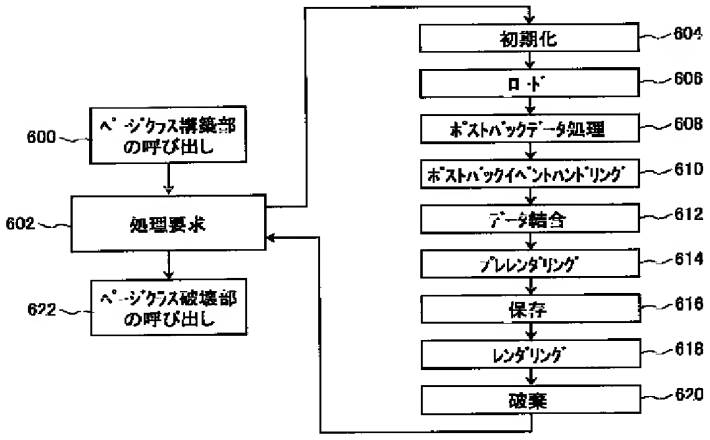
【図２】


クライアント
200
ASP+ファイルを特定するURL
でHTTPリクエストをサーバ
に送信
サーバ
202
HTTPリクエストを受信
203
ASP+ファイルの読み出し
204
特定のASP+ファイルの制御
オブジェクト階層を生成
206
サーバ側制御オブジェクトを
用いてクライアント側ユーザ
インタフェース要素を処理
208
クライアント側ユーザインタフェース
要素のHTMLデータを
レンダリング
210
HTTPレスポンスでクライアントに
HTMLを送信
212
制御オブジェクト階層を破棄
214
新たなページ用のHTMLデータ
でHTTPレスポンスを受信
216
クライアント側ユーザインタフェース
要素 を記述するHTML
データでブラウザに新たな
ページを表示

【図３】


300
ウェブサーバ
304
306
310
302
ASP+ファイル
ページファクトリ
制御クラス
ライブラリ
308
312
314
HTTP
パイプ
ライン
ページ
オブジェクト
330
アプリケーション
コンポーネント
テキスト
ボックス
オブジェクト
ボタン
オブジェクト
ボタン
オブジェクト
324
318
320
322
316

【図６】

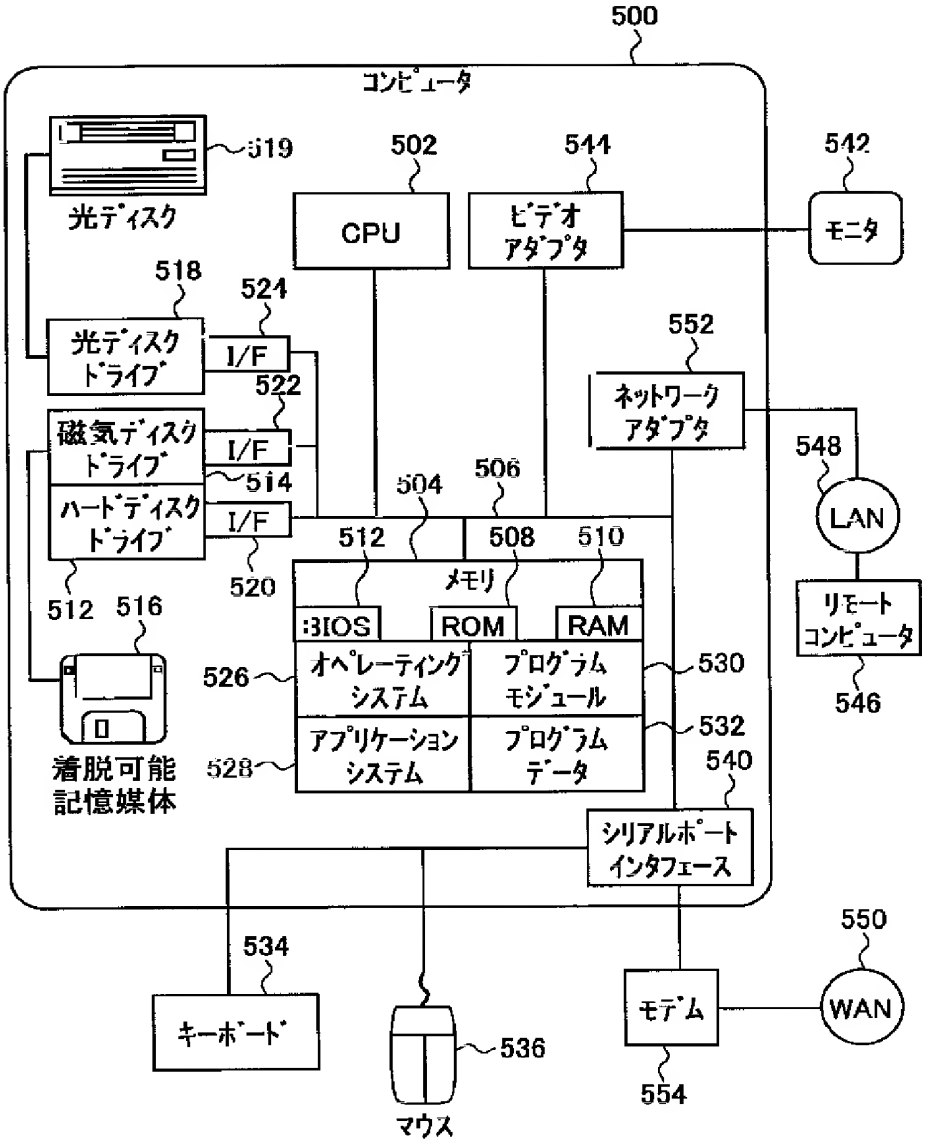

600
ページクラス構築部
の呼び出し
602
処理要求
622
ページクラス破壊部
の呼び出し
初期化
604
ロード
606
ポストバックデータ処理
608
ポストバックイベントハンドリング
610
データ結合
612
プレレンダリング
614
保存
616
レンダリング
618
破棄
620

【図4】

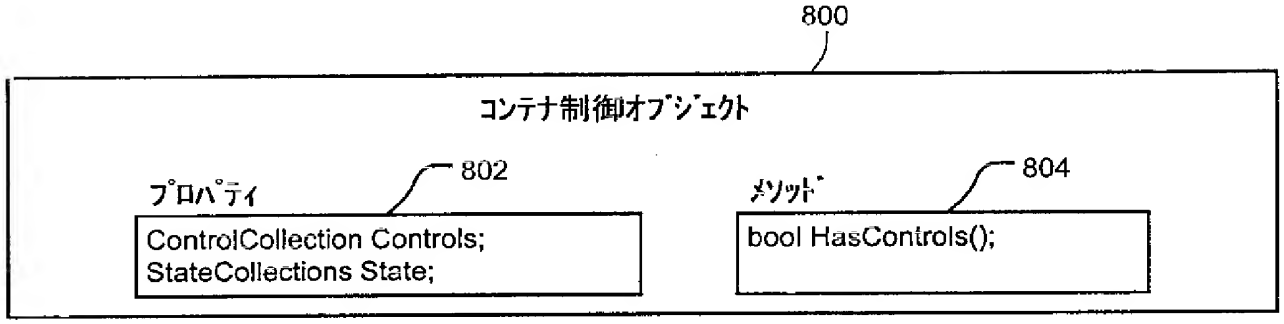
400

```
1  <%@ Page Language="VB" Description="Simple Sample Page" Errorpage="ErrorPage.aspx" %>
2  <html>
3      <script runat=server>
4          Sub AddButton_Click(ByVal Source as Object, By Val E as Event Args)
5              Message.Text = "Add" & UserName.Text
6          End Sub
7          Sub DeleteButton_Click(ByVal Source as Object, By Val E as Event Args)
8              Message.Text = "Delete" & UserName.Text
9          End Sub
10     </script>
11     <body>
12         <form runat="server">
13             User Name:      <input type="Text" id="UserName" runat=server>
14             <br>
15             <button id="AddButton" value="ADD" OnClick="AddButton_Click" runat=server>
16             <button id="DeleteButton" value="DELETE" OnClick="DeleteButton_Click" runat=server>
17             <br><br>
18             <span id="Message" runat=server> </span>
19         </form>
20     </body>
21 </html>
```

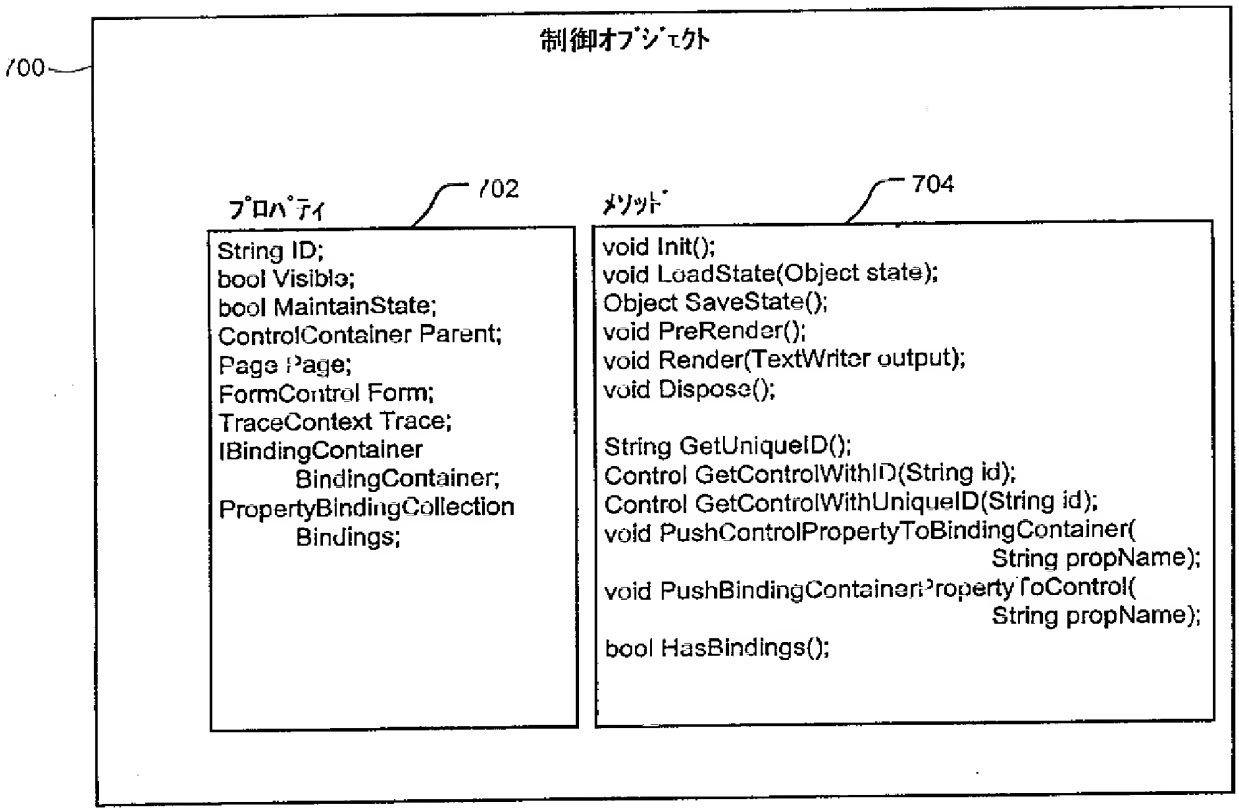
【図５】


500
コンピュータ
519
光ディスク
502
CPU
544
ビデオアダプタ
542
モニタ
518
光ディスクドライブ
524
I/F
522
磁気ディスクドライブ
I/F
514
ハードディスクドライブ
I/F
520
552
ネットワークアダプタ
548
LAN
リモートコンピュータ
546
504
506
512
508
510
メモリ
BIOS
ROM
RAM
526
オペレーティングシステム
プログラムモジュール
530
528
アプリケーションシステム
プログラムデータ
532
540
シリアルポートインタフェース
512
516
着脱可能記憶媒体
534
キーボード
536
マウス
モデム
554
550
WAN

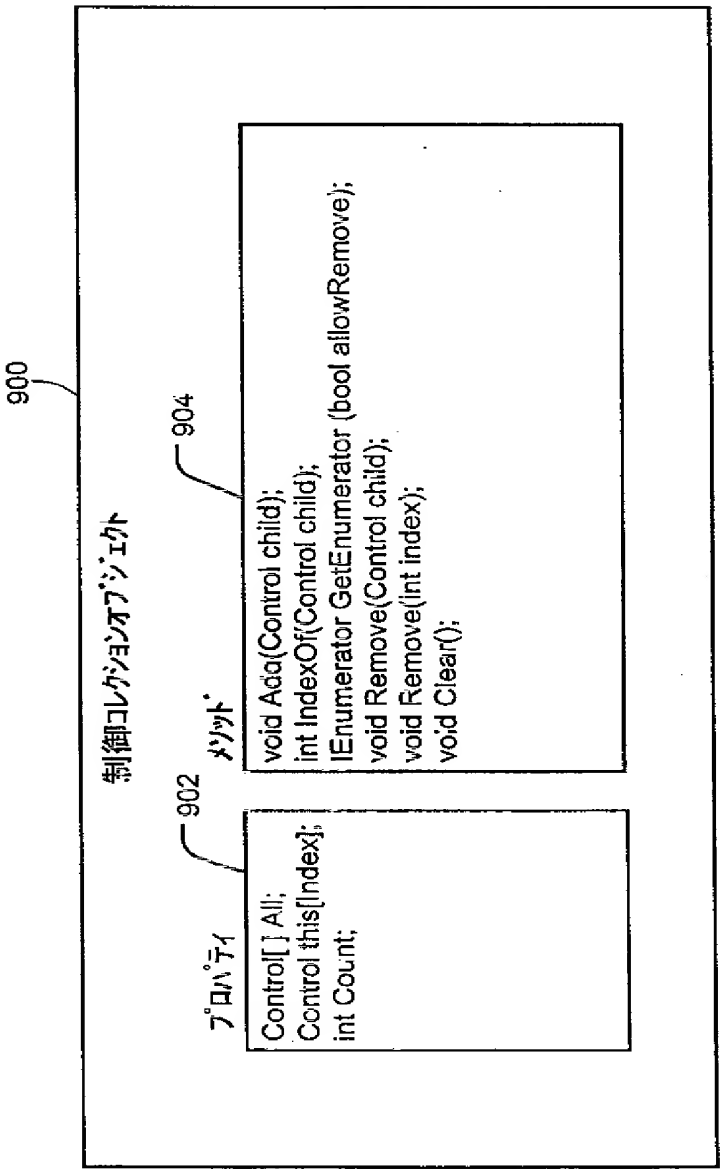
【図８】


800
コンテナ制御オブジェクト
プロパティ
802
ControlCollection Controls;
StateCollections State;
メソッド
804
bool HasControls();

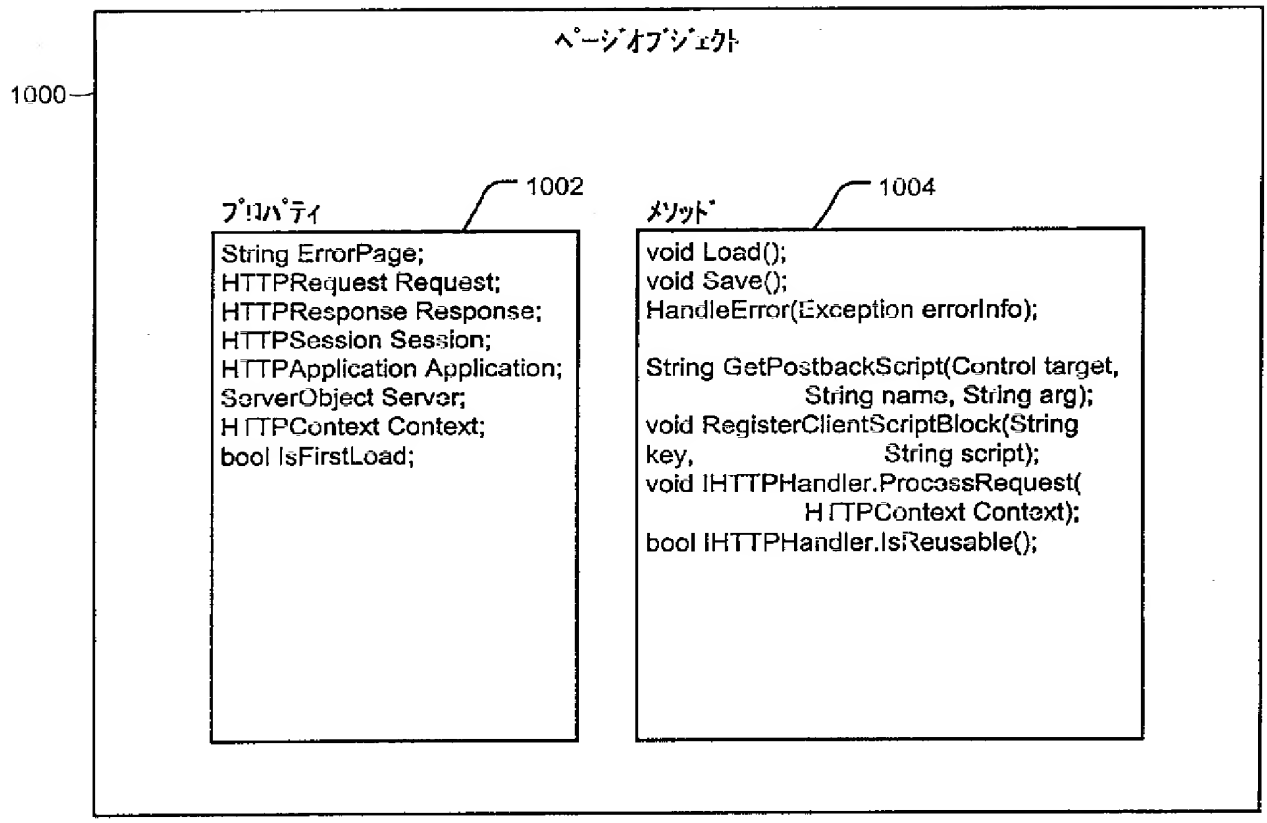
【図７】


制御オブジェクト
700
プロパティ
702
String ID;
bool Visible;
bool MaintainState;
ControlContainer Parent;
Page Page;
FormControl Form;
TraceContext Trace;
IBindingContainer
BindingContainer;
PropertyBindingCollection
Bindings;
メソッド
704
void Init();
void LoadState(Object state);
Object SaveState();
void PreRender();
void Render(TextWriter output);
void Dispose();
String GetUniqueID();
Control GetControlWithID(String id);
Control GetControlWithUniqueID(String id);
void PushControlPropertyToBindingContainer(
String propName);
void PushBindingContainerPropertyToControl(
String propName);
bool HasBindings();

【図9】


900
制御コレクションオブジェクト
902
プロパティ
Control[ ] All;
Control this[Index];
int Count;
904
メソッド
void Add(Control child);
int IndexOf(Control child);
IEnumerator GetEnumerator (bool allowRemove);
void Remove(Control child);
void Remove(int index);
void Clear();

【図１０】



【手続補正書】
【提出日】平成１３年６月１２日（２００１．６．１２）
【手続補正１】
【補正対象書類名】明細書
【補正対象項目名】００９６
【補正方法】変更
【補正内容】
【００９６】
【発明の効果】以上のように本発明によれば、クライアント側ユーザインタフェース要素の処理および生成を管理するサーバ側制御オブジェクトフレームワークを提供することができることから、クライアント側ユーザインタフェース要素および関連機能に関するサーバ側処理のカプセル化を図ることができ、開発者が最小のプログラミングでウェブページを動的に作成および処理することができる開発フレームワークを提供することが可能となる。
【手続補正２】
【補正対象書類名】明細書
【補正対象項目名】符号の説明
【補正方法】変更
【補正内容】
【符号の説明】
１００　クライアント
１０２　ブラウザ
１０４　ウェブページ
１０６　ユーザ名
１０８　追加
１１０　削除
１１２　HTTPレスポンス
１１４　HTTPリクエスト
１１６、３００　ウェブサーバ
１１８、３０４　HTTPパイプライン
１２０　ハンドラ
１２２　静的コンテンツファイル
１２４　動的コンテンツファイル
１２６　サーバ側制御クラスライブラリ
１２８　クライアント側制御クラスライブラリ
１３０、３３０　アプリケーションコンポーネント
３０８　ページファクトリ
３１０　ASP+ファイル
３１２　制御クラスライブラリ

３１４　ページオブジェクト
３１８　テキストボックスオブジェクト
３２０、３２２　ボタンオブジェクト
５００　コンピュータ
５０４　メモリ
５１２　ハードディスクドライブ
５１４　磁気ディスクドライブ
５１６　着脱可能記憶媒体
５１８　光ディスクドライブ
５１９　光ディスク
５２０、５２２、５２４　Ｉ／Ｆ
５２６　オペレーティングシステム
５２８　アプリケーションプログラム
５３０　プログラムモジュール
５３２　プログラムデータ
５３４　キーボード
５３６　マウス
５４０　シリアルポートインタフェース
５４２　モニタ
５４６　リモートコンピュータ
５５２　ネットワークアダプタ
５５４　モデム
７００　制御オブジェクト
７０２、８０２、９０２、１００２　プロパティ
７０４、８０４、９０４、１００４　メソッド
８００　コンテナ制御オブジェクト
９００　制御コレクションオブジェクト
１０００　ページオブジェクト

フロントページの続き

(72)発明者　クーパー、ケネス　ビー.
アメリカ合衆国、98199　ワシントン州　シアトル、ウエスト　ブレイン　3410
(72)発明者　ギャスリー、スコット　ディー.
アメリカ合衆国、98052　ワシントン州　レッドモンド、アパートメント　ユー2102、エヌイー　90ス　ストリート　17786
(72)発明者　エボ、デイビッド　エス.
アメリカ合衆国、98052　ワシントン州　レッドモンド、アパートメント　ジェイ139、アボンデイル　ロード　10909
(72)発明者　アンダース、マーク　ティー.
アメリカ合衆国、98007　ワシントン州　ベルビュー、エヌイー　12ス　プレイス　14425
(72)発明者　ピータース、テッド　エー.
アメリカ合衆国、98119　ワシントン州　シアトル、８ス　アベニュー　ウエスト　2431
(72)発明者　ミレット、スティーブン　ジェイ.
アメリカ合衆国、98026　ワシントン州　エドモンズ、オリンピック　ビュー　ドライブ　8412

Ｆターム(参考)　5B076 DA01 DA06 DF08

【外国語明細書】

1 Title of Invention

SERVER-SIDE CONTROL OBJECTS FOR PROCESSING CLIENT-SIDE USER INTERFACE ELEMENTS

2 Claims

1. A method for processing a client side user interface element incorporated in a web page displayed on a client, the method comprising:

receiving a request referencing a resource;

reading a declaration from the resource;

generating a server-side control object programmed based on the declaration to provide functionality of the client side user interface element;

processing the client-side user interface element using the server-side control object; and

generating authoring language data from the server-side control object for incorporating the client-side user interface element in the web page, subsequent to the processing operation.

2. The method of claim 1 wherein the resource is a server-side declaration datastore.

3. The method of claim 1 further comprising:

transmitting the authoring language data to the client; and

terminating the server-side control object, subsequent to the operation of generating authoring language data.

4. The method of claim 1 wherein the processing operation comprises:

handling postback data received from the user interface element using the server-side control object.

5. The method of claim 1 wherein the processing operation comprises:

handling postback events received from the user interface element using the server-side control object.

6. The method of claim 1 wherein the processing operation comprises:

saving a viewstate of the server-side control object; and

transmitting the viewstate to the client.

7. The method of claim 1 wherein the processing operation comprises:

receiving a viewstate of the server-side control object from the client, the viewstate corresponding to the state of the server-side control object from a previous request; and

loading the viewstate to the server side control object.

8. A computer data signal embodied in a carrier wave by a computing system and encoding a computer program for executing a computer process processing a client-side user interface element incorporated in a web page displayed on a client, the computer process comprising:

receiving a request referencing a resource;

reading a declaration from the resource;

generating a server-side control object programmed based on the declaration to provide functionality of the client-side user interface element;

processing the client-side user interface element using the server-side control object; and

generating authoring language data from the server-side control object for incorporating the client-side user interface element in the web page, subsequent to the processing operation.

9. A computer program storage medium readable by a computer system and encoding a computer program for executing a computer process processing a client side user interface element incorporated in a web page displayed on a client, the computer process comprising:

receiving a request referencing a resource;

reading a declaration from the resource;

generating a server-side control object programmed based on the declaration to provide functionality of the client-side user interface element;

processing the client-side user interface element using the server-side control object; and

generating authoring language data from the server side control object for incorporating the client-side user interface element in the web page, subsequent to the processing operation.

10. A hierarchy of server-side control objects, executable by a computer, for processing one or more client-side user interface elements incorporated in a web page displayed on a client, the hierarchy of server-side control objects comprising:

one or more server-side child objects corresponding to the one or more client side user interface elements, each server-side child object being adapted to handle postback input received from the client and being adapted to generate authoring language data for display of the client-side user interface element on the client;

at least one hierarchical identifier received from the client in association with the postback input specifying one of the server-side child objects; and

a server-side page object hierarchically related to the one or more server-side child objects and receiving the postback input for distribution to the one of the server-side child objects based on the hierarchical identifier.

11. The hierarchy of server-side control objects of claim 10 wherein the page object hierarchically contains the one or more server-side child objects.

12. The hierarchy of server-side control objects of claim 10 wherein the page object is created responsive to a request from the client and remains in existence until after generation of the authoring language data for the web page.

13. The hierarchy of server-side control objects of claim 10 wherein each server-side child object is created responsive to an access to the server-side control object specified by the hierarchical identifier and remains in existence until after generation of the authoring language data for a corresponding client-side user interface element on the client.

14. A computer program storage medium readable by a computing system and encoding a computer program for executing a computer process that processes one or more client-side user interface elements incorporated in a web page on a client, the computer process comprising:

inputting one or more declarations from a server-side declaration datastore;

generating a hierarchy of server-side control objects programmed based on the declarations to provide functionality of the client-side user interface elements;

processing the client-side user interface elements using the hierarchy of server-side control objects; and

generating authoring language data from the hierarchy of server-side control objects for incorporating the client-side user interface elements in the web page.

15. The computer program storage medium of claim 14 wherein the operation of generating a hierarchy of server-side control objects comprises:

generating a server-side container control object corresponding to a client-side user interface container element on the web page; and

generating one or more server-side children control objects corresponding to one or more client-side user interface child elements contained in the client-side user interface container element.

16. The computer program storage medium of claim 15 wherein the operation of processing the client-side user interface elements comprises:

receiving a unique hierarchical identifier referencing one or more of the server-side control objects in the hierarchy of server-side control objects;

receiving postback input information associated with the unique hierarchical identifier;

resolving the unique hierarchical identifier to identify a referenced server-side control object;

passing the postback input information to the referenced server-side control object; and

processing the postback input information using the referenced server-side control object.

17. The computer program storage medium of claim 14 wherein the operation of processing the client-side user interface elements comprises:

traversing the hierarchy of the server-side control objects to invoke individual processing by one or more of the server-side control objects.

18. The computer program storage medium of claim 14 wherein the operation of processing the client side user interface elements comprises:

registering one or more of the server -side control objects to handle a control

object event generated by one of the server-side control objects;

generating the control object event from one or more of the server-side control objects; and

handling the control object event using one or more of the server-side control objects registered to handle the control object event.

19. The computer program storage medium of claim 14 wherein the operation of processing the client-side user interface elements comprises:

loading a first viewstate to one or more of the server-side control objects;

handling postback data using one or more of the server-side control objects, subsequent to the operation of loading the first viewstate;

handling postback events using one of more of the server-side control objects, subsequent to the operation of handling postback data; and

saving a second viewstate from one or more of the server side control objects, subsequent to the operation of handling postback events.

20. The computer program storage medium of claim 19 wherein the operation of processing the client-side user interface elements further comprises:

resolving a databinding relationship established between a server side control objects and a server-side datastore, subsequent to the operation of handling postback events.

21. A method for processing at least one client-side user interface element incorporated in a web page displayed on a client, the method comprising:

reading a declaration from a resource;

generating a plurality of concurrently existing server-side control objects that logically correspond to the client-side user interface element, based on the declaration;

processing the client-side user interface element using the concurrently existing server-side control objects; and

generating authoring language data from the concurrently existing server side control objects for incorporating the client side user interface element in the web page, subsequent to the processing operation.

22. The method of claim 21 wherein the operation of generating a plurality of concurrently existing server-side control objects comprises:

generating a first server-side control object that corresponds to a first client-side user interface element; and

generating a second server-side control object that corresponds to a first client-side user interface element, such that the first and the second server side control objects exist concurrently.

23. The method of claim 21 wherein the processing operation comprises: raising a server-side event associated with a first server-side control object.

24. The method of claim 23 wherein the processing operation further comprises: handling the server-side event using a second server-side control object.

25. The method of claim 23 wherein the processing operation further comprises: handling the server-side event using a non-user-interface server component.

26. The method of claim 21 wherein the operation of generating a plurality of concurrently existing server-side control objects comprises:

generating a first server-side control object that corresponds to a client-side user interface element; and

generating a second server-side control object that corresponds to the client-side user interface element, such that the first and the second server-side control objects exist concurrently.

27. A computer data signal embodied in a carrier wave by a computing system and encoding a computer program for executing a computer process processing at least one client-side user interface element incorporated in a web page displayed on a client, the computer process comprising:

reading a declaration from a resource;

generating a plurality of concurrently existing server side control objects that logically correspond to the client-side user interface element, based on the declaration;

processing the client-side user interface element using the concurrently existing server-side control objects; and

generating authoring language data from the concurrently existing server-side control objects for incorporating the client-side user interface element in the web page, subsequent to the processing operation.

28. A computer program storage medium readable by a computing system and encoding a computer program for executing a computer process processing at least one client-side user interface element incorporated in a web page displayed on a client, the computer process comprising:

reading a declaration from a resource;

generating a plurality of concurrently existing server-side control objects that logically correspond to the client-side user interface element, based on the declaration;

processing the client-side user interface element using the concurrently existing server-side control objects; and

generating authoring language data from the concurrently existing server-side control objects for incorporating the client-side user interface element in the web page, subsequent to the processing operation.

3 Detailed Description of Invention

Technical Field

The invention relates generally to a web server framework, and more particularly to server-side control objects that process client-side user interface elements of a web page.

Background of the Invention

A typical web browser receives data from a web server that defines the appearance and rudimentary behavior of a web page for display on a client system. In a typical scenario, a user specifies a Uniform Resource Locator ("URL"), a global address of a resource on the World Wide Web, to access a desired web site. Generally, the term "resource" refers to data or routines that can be accessed by a program. An example URL is "http://www.microsoft.com/ms.htm". The first part of the example URL indicates a given protocol (i.e., "http") to be used in the communication. The second part specifies the domain name (i.e., "www.microsoft.com") where the resource is located. The third part specifies the resource (i.e., a file called "ms.htm") within the domain. Accordingly, a browser generates an HTTP (HyperText Transport Protocol) request associated with the example URL to retrieve the data associated with ms.htm file within the www.microsoft.com domain. A web server hosting the www.microsoft.com site receives the HTTP request and returns the requested web page or resource in an HTTP response to the client system for display in the browser.

The "ms.htm" file of the example above includes static HTML (HyperText Markup Language) code. HTML is a plain-text authoring language used to create documents (e.g., web pages) on the World Wide Web. As such, an HTML file can be retrieved from a web server and displayed as a web page in a browser to present the rich graphical experience that users have come to expect while viewing information from the Internet. Using HTML, a developer can, for example, specify formatted text, lists, forms, tables, hypertext links, inline images and sounds, and

background graphics for display in the browser. An HTML file, however, is a static file that does not inherently support dynamic generation of web page content.

In some circumstances, a web page may need to display dynamic content in a browser, such as a changing stock price or traffic information. In such situations, a server-side application program is typically developed to obtain the dynamic data and format it into HTML that is sent to the browser for display in a web page as the web page is updated.

Additionally, these same server side applications programs may be used in situations where the data is not strictly dynamic but where there are so many different values that may be displayed in a static web page that it would be impractical to create the required number of static web pages. For example, a trip planning page might display two calendars: one calendar for departure date and one calendar for return date. Rather than developing hundreds of static pages with every possible pair of calendar combinations, a server-side application program can dynamically generate the appropriate static page with the appropriate calendars displayed.

Many web pages allow the user to interact with the page displayed in the browser by selecting visual elements of the page. For example, in the above mentioned trip planning page, the calendar might allow the user to interact with it, by clicking on a day to select that date, or clicking on an icon to go forward or backward a month. In existing solutions, a browser will submit an HTTP request back to the server side application program. The HTTP request can include parameters encoded in the query string, as form post variables, or some other data format to describe client-side events or data (e.g., which control the user clicked on). For example, the parameters might include the date that the user selected in one calendar, along with the date currently displayed in the other calendar.

The communication of events and data back to the server is called a "post back" because the browser typically sends the request using an HTTP POST request. The server-side application program can process the HTTP request and generate the appropriate HTML code for web page with a newly computed calendar to reflect the user's action for transmission to the client in an HTTP response. Thereafter, the resulting document is transmitted to a client system in an HTTP response, where it is displayed in the browser as a web page that shows the updated calendars.

Developing a server-side application program can be a complex task requiring not only familiarity with normal HTML coding that is used to layout a web page, but also with the programming basics, including one or more programming languages (e.g., C++, Perl, Visual Basic, or Jscript), and the HTTP protocol, including how data is sent between the browser and the server. Web page designers, on the other hand, are frequently graphics designers and editors, who may lack programming experience. Furthermore, simplifying complex web page development can speed the development of new web content by any developer.

Generally, development of a custom server-side application program also requires tremendous effort, so much, in fact, that developers are often disinclined to attempt it. Not only must a developer understand the HTML code that must be generated to display a desired web page, but the developer must understand how user interaction and client data from the web page will result in post back operations. It is desirable, therefore, to provide a development framework that allows a developer to dynamically create and process a web page with minimal programming.

One approach to minimize the programming requirements of dynamic web page generation has been the Active Server Page (ASP) framework, provided by Microsoft Corporation. An ASP resource typically includes Visual Basic or Jscript code, for example, to process an HTTP request that specifies the ASP resource as the desired resource and, thereafter, to generate the resulting HTML code in a HTTP response to the client. Furthermore, an ASP resource may reference pre-developed or third party client-side library components (e.g., client-side ACTIVEX controls) to ease a given application programming effort. However, in the current server-side application frameworks, the programming required to dynamically manage client-side user interface elements (e.g., text boxes, list boxes, buttons, hypertext links, images, sounds, etc.) within server-side applications can still require sophisticated programming skills and considerable effort. An unanswered problem exists in properly encapsulating programming required to process user interface elements, including handling postback events, so as to allow the web page developer to focus on other aspects of the web page.

Summary of the Invention

In accordance with the present invention, the above and other problems are solved by providing a server-side control object framework to manage the processing and generating of client-side user interface elements. Furthermore, a hierarchy of server-side control objects can cooperate to generate the resulting authoring language code, such as standard HTML, for display of a web page on a client. The client can be, for example, any browser that supports standard HTML or another authoring language. The operation of processing the client-side user interface element may include one or more of a postback event handling operation, a postback data handling operation, a data binding operation, or a state management operation relating to the state of a server-side control object.

A great utility of an embodiment of the present invention lies in improved encapsulation of server-side processing of client-side user interface elements (e.g., input received from user interface elements and output used to generate user interface elements) and related functionality. One or more server-side control objects may be generated to logically correspond to one or more user interface elements. For example, given a user interface element representing a month on a calendar display, a hierarchy of server-side control objects may be generated corresponding to the calendar display and its various sub-elements. In one configuration, a "month" control object can hierarchically contain multiple "week" control objects, wherein each "week" control object hierarchically contains seven "day" control objects.

In addition, in an embodiment of the present invention, the server-side control objects can cooperate to process the logically corresponding user interface elements. This advantage results, in part, from the concurrent existence of multiple server-side control objects during the processing of a client request and the generation of a response. For example, upon detecting a first postback event received from the client (e.g., a click on a "next month" button element on a calendar display), one control object can raise a second event (e.g., selection of the next month for display by the "month" control object) that is thereafter detected and processed by one or more concurrently existing control objects. This cooperation provides encapsulation of complex control interaction within the server-side control

objects themselves, thereby minimizing the custom event handling required of the web page developer.

A method and computer program product that process a client side user interface element incorporated in a web page displayed on a client are provided. A request referencing a server-side declaration datastore is received. A declaration is inputted from the server-side declaration datastore. A server-side control object is generated and programmed to provide functionality of the user interface element based on the declaration. The user interface element is processed using the server-side control object. Authoring language data is generated from the server-side control object for displaying the user interface element in the web page.

A hierarchy of server-side control objects, executable by a computer, for processing one or more client side user interface elements incorporated in a web page displayed on a client is provided. One or more server-side child objects correspond to the one or more client-side user interface elements. Each server-side child object handles input received from the client and generates authoring language data for display of the client-side user interface element on the client. At least one hierarchical identifier is received from the client in association with the input specifying one of the server-side child objects. A server-side page object contains the server-side child objects and receives the input for distribution to one of the server-side child objects in accordance with the hierarchical identifier.

Detailed Description of the Invention

An embodiment of the present invention includes a server-side control object for processing and generating a client-side user interface element for display on a web page. Furthermore, a hierarchy of server-side control objects can cooperate to generate the resulting authoring language code, such as standard HTML, for display of a web page on a client. The client can be, for example, any browser that supports standard HTML or another authoring language. In an embodiment of the present invention, server-side control objects logically correspond to client-side user interface elements and generate at a server the authoring language code to be used by a client-side browser to display and process a web page. The operation of processing the client-side user interface element may include one or more of a postback event handling operation, a postback data handling operation, a data binding operation, and a state management operation. The state management operation relates to the state of a server-side control object.

FIG. 1 illustrates a web server for dynamically generating web page content for display on a client in an embodiment of the present invention. A client 100 executes a browser 102 that displays a web page 104 on a display device of the client 100. The client 100 may include a client computer system having a display device, such as a video monitor. An "INTERNET EXPLORER" browser, marketed

by Microsoft Corporation, is an example of a browser 102 in an embodiment of the present invention. Other exemplary browsers include without limitation "NETSCAPE NAVIGATOR" and "MOSAIC". The exemplary web page 104 incorporates a text box control 106 and two button controls 108 and 110. The browser 102 may receive HTML code in the HTTP response 112 from a web server 116 and displays the web page as described by the HTML code. Although HTML is described with reference to one embodiment, other authoring languages, including without limitation SGML (Standard Generalized Markup Language), XML (eXtensible Markup Language), and WML (Wireless Markup Language), which is an XML based markup language, designed for specifying the content and user interfaces of narrowband wireless devices, such as pagers and cellular phones, are contemplated within the scope of the present invention. Furthermore, although standard HTML 3.2 is primarily disclosed herein, any version of HTML is supportable within the scope of the present invention.

The communications between the client 100 and the web server 116 may be conducted using a sequence of HTTP requests 114 and HTTP responses 112. Although HTTP is described with reference to one embodiment, other transport protocols, including without limitation S-HTTP, are contemplated within the scope of the present invention. On the web server 116, an HTTP pipeline module 118 receives an HTTP request 114, resolves the URL, and invokes an appropriate handler 120 for processing the request. In an embodiment of the present invention, a plurality of handlers 120 to handle different types of resources are provided on the web server 116.

For example, if the URL specifies a static content resource 122, such as an HTML file, a handler 120 accesses the static content resource 122 and passes the static content resource 122 back through the HTTP pipeline 118 for communication to the client 100 in an HTTP response 112. Alternatively, in an embodiment of the present invention, if the URL specifies a dynamic content resource 124, such as an ASP+ resource, a handler 120 accesses the dynamic content resource 124, processes the contents of the dynamic content resource 124, and generates the resulting HTML code for the web page 104. In an embodiment of the present invention, the resulting HTML code includes standard HTML 3.2 code. Generally, a dynamic content resource is a server-side declaration datastore (e.g., an ASP+ resource) that can be

used to dynamically generate the authoring language code that describes a web page to be displayed on a client. The HTML code for the web page is then passed through the HTTP pipeline 118 for communication to the client 100 in an HTTP response 112.

During its processing, a handler 120 can also access libraries of pre-developed or third party code to simplify the development effort. One such library is a server-side class control library 126, from which the handler 120 can instantiate server-side control objects for processing user interface elements and generating the resultant HTML data for display of a web page. In an embodiment of the present invention, one or more server-side control objects map to one or more user interface elements, visible or hidden, on the web page described in the dynamic content resource 124.

A second library, in contrast, is a client-side control class library 128, such as a library including "ACTIVEX" components from Microsoft Corporation. An "ACTIVEX" control is a COM ("Component Object Model") object that follows certain standards in how it interacts with its client and other components. A client-side "ACTIVEX" control, for example, is a COM-based component that can be automatically downloaded to a client and executed by a web browser on the client. Server side ACTIVEX components (not shown) are COM-based components that may be implemented on a server to perform a variety of server side functions, such as providing the server-side functionality of a stock price look-up application or database component. A more detailed discussion of ACTIVEX can be found in "Understanding ACTIVEX and OLE", David Chappell, Microsoft Press, 1996.

In contrast to "ACTIVEX" controls, a server-side control object in an embodiment of the present invention, being specified in a dynamic content resource 124, logically corresponds to a user interface element that is displayed on the client. The server side control object can also generate valid HTML code that can include, for example, an HTML tag and a locator referencing a given client-side "ACTIVEX" control. If the browser already has the code for the client-side "ACTIVEX" control within its storage system, the browser executes the "ACTIVEX" control within the web page on the client. Otherwise, the browser downloads the code for the "ACTIVEX" control from the resource specified by the locator and then executes the "ACTIVEX" control within the web page on the client.

A server-side control object in an embodiment of the present invention can also raise events to a server-side "ACTIVEX" object used to implement a stock look-up application on the server.

A handler 120 also has access to one or more non-user-interface server components 130 that execute on the web server 116 or on another accessible web server. A non user-interface server component 130, such as a stock price look-up application or database component, may be referenced in or associated with a dynamic content resource 124 that is processed by a handler 120. Server-side events raised by the control objects declared in the dynamic content resource 124 may be processed by server-side code, which calls appropriate methods in the non-user-interface server component 130. As a result, the processing provided by the server-side control objects simplifies the programming of the non-user-interface server component 130 by encapsulating the processing and generation of the user interface elements of a web page, which allows the developer of the non-user-interface server component 130 to concentrate on the specific functionality of the application, rather than on user interface issues.

FIG. 2 illustrates a flow diagram of operations for processing and generating client-side user interface elements using server-side control objects in an embodiment of the present invention. In operation 200, the client transmits an HTTP request to the server. The HTTP request includes a URL that specifies a resource, such as an ASP+ resource. In operation 202, the server receives the HTTP request and invokes the appropriate handler for processing the specified resource. The ASP+ resource is read in operation 203. Operation 204 generates a server-side control object hierarchy based on the contents of the specified dynamic content resource (e.g., the ASP+ resource).

In operation 206, the server-side control objects of the control object hierarchy perform one or more of the following operations: postback event handling, postback data handling, state management, and data binding. Postback events and data (collectively "postback input") from user interface elements are communicated from the client to the server for processing. A postback event, for example, may include without limitation a "mouse click" event from a client-side button element or a "data change" event from a client-side textbox element that is communicated to the server. Postback data, for example, may include without

limitation text entered by a user in a text box element or an index of an item selected from a drop-down box. A postback operation, however, may result from other events, and not just from user interaction.

In operation 208, each server-side control object in the hierarchy is called to generate (or render) data, such as HTML code, for display of client-side user interface elements in the web page. Note that, although the term "render" may be used to describe the operation of displaying graphics on a user interface, the term "render" is also used herein to describe the operation of generating authoring language data that can be interpreted by client-application, such as a browser, for display and client-side functionality. A more detailed discussion of the processing operation 206 and the rendering operation 208 is provided in association with FIG. 6. In one embodiment, calls to render() methods in individual control objects are performed using a tree traversal sequence. That is, a call to the render() method of a page object results in recursive traversal throughout appropriate server-side control objects in the hierarchy. Alternative methods for calling the render() methods for appropriate control objects may also be employed, including an event signaling or object registration approach. The parentheses designate the "render()" label as indicating a method, as compared to a data value.

In an embodiment of the present invention, the actual creation of the individual server-side control objects may be deferred until the server-side control object is accessed (such as when handling postback input, loading a state, rendering HTML code from the control object, etc.) in operations 206 or 208. If a server-side control object is never accessed for a given request, deferred control object creation optimizes server processing by eliminating an unnecessary object creation operation.

Operation 210 transmits the HTML code to the client in an HTTP response. In operation 214, the client receives the HTML code associated with a new web page to be displayed. In operation 216, the client system incorporates (e.g., displays) the user interface elements of the new page in accordance with the HTML code received from the HTTP response. It should be understood, however, that incorporation of a user-interface element may include non-display operations, such as providing audio or tactile output, reading and writing to memory, controlling the operation of scripts, etc. In operation 212, the server-side control object hierarchy is

terminated. In an embodiment of the present invention, server-side control objects in the hierarchy are created in response to an HTTP request referencing an associated ASP+ resource, and destroyed subsequent to the rendering of authoring language data (e.g., HTML data). In an alternative embodiment, operation 212 may be performed after operation 208 and before operation 210.

FIG. 3 illustrates exemplary modules in a web server used in an embodiment of the present invention. The web server 300 receives an HTTP request 302 into the HTTP pipeline 304. The HTTP pipeline 304 may include various modules, such as modules for logging of web page statistics, user authentication, user authorization, and output caching of web pages. Each incoming HTTP request 302 received by the web server 300 is ultimately processed by a specific instance of an IHTTPHandler class (shown as handler 306). The handler 306 resolves the URL request and invokes an appropriate handler factory (e.g., a page factory module 308).

In FIG. 3, a page factory module 308 associated with the ASP+ resource 310 is invoked to handle the instantiation and configuration of the ASP+ resource 310. In one embodiment, an ASP+ resource can be identified by designating a particular suffix (or a file extension such as ".aspx") with the resource. When a request for a given ASP+ resource is first received by the page factory module 308, the page factory module 308 searches the file system for the appropriate file (e.g., the .aspx file 310). The file may contain text (e.g., authoring language data) or another data format (e.g., byte-code data or encoded data) that may later be interpreted or accessed by the server to service the request. If the physical file exists, the page factory module 308 opens the file and reads the file into memory. If the file cannot be found, the page factory module 308 returns an appropriate "file not found" error message.

After reading the ASP+ resource 310 into memory, the page factory module 308 processes the file content to build a data model of the page (e.g., lists of script blocks, directives, static text regions, hierarchical server-side control objects, server-side control properties, etc.). The data model is used to generate a source listing of a new object class, such as a COM+ ("Component Object Model+") class, that extends the page base class. The page base class includes code that defines the structure, properties, and functionality of a basic page object. In an embodiment of the present invention, the source listing is then dynamically compiled into an

intermediate language and later Just-In-Time compiled into platform native instructions (e.g., X86, Alpha, etc.). An intermediate language may include general or custom-built language code, such as COM+ IL code, Java bytecodes, Modula 3 code, SmallTalk code, and Visual Basic code. In an alternative embodiment, the intermediate language operations may be omitted, so that the native instructions are generated directly from the source listing or the source file (e.g., the ASP+ resource 310). A control class library 312 may be accessed by the page factory module 308 to obtain predefined server-side control classes used in the generation of the control object hierarchy.

The page factory module 308 outputs a page object 314, which is a server-side control object that corresponds to the web page 104 of FIG. 1. The page object 314 and its children (i.e., a text box object 318, a button object 320, and another button object 322) comprise an exemplary control object hierarchy 316. Other exemplary control objects are also contemplated in accordance with the present invention, including without limitation objects corresponding to the HTML controls in Table 1, as well as custom control objects. The page object 314 corresponds to the web page 104 of FIG. 1. The text box object 318 corresponds to the text box 106 in FIG. 1. Likewise, the button object 320 corresponds to the add button 108 in FIG. 1, and the button object 322 corresponds to the delete button 110 in FIG. 1. The page object 314 is hierarchically related to other control objects on the server. In one embodiment, a page object is a container object that hierarchically contains its children control objects. In an alternative embodiment, other forms of hierarchical relation may be employed, including a dependency relationship. In a more complex control object hierarchy with multiple levels of children, a child object can be a container object for other child objects.

In the illustrated embodiment, the control objects in the control object hierarchy 316 are created and executed on the server 300, and each server-side control object logically corresponds to a corresponding user interface element on the client. The server-side control objects also cooperate to handle postback input from the HTTP request 302, to manage the states of server-side control objects, to perform data binding with server-side databases, and to generate authoring language data (e.g., HTML code) used to display a resulting web page at the client. The resulting authoring language data is generated (i.e., rendered) from the server-side

control object hierarchy 316 and transmitted to the client in an HTTP response 324. For example, resulting HTML code may embody any valid HTML construct and may reference ACTIVEX-type controls, JAVA applets, scripts, and any other web resources that yield client-side user interface elements (e.g., control buttons, text boxes, etc.) when processed by a browser.

By virtue of declarations made in the ASP+ resource 310, server-side control objects may also access one or more non-user-interface server components 330 to provide interaction between the non-user-interface server component 330 and client-side user interface elements. For example, in response to postback input, server-side control objects can raise server-side events to the non-user interface server components registered for those events. In this manner the non-user-interface server component 330 can interact with the user through user interface elements without programming the code required to display and process these elements.

FIG. 4 illustrates contents of an exemplary dynamic content resource in an embodiment of the present invention. In the illustrated embodiment, the file 400 contains plain text declarations in an exemplary dynamic content resource format (e.g., ASP+). Each declaration provides instructions to a page compiler that reads the file 400, creates and invokes the appropriate server-side control objects to process the client side user interface element, and ultimately combines the rendered HTML code for transmission to the client in an HTTP response. As such, a declaration describes or references functionality of a client-side user interface element, which is implemented by a server-side control object. The server side control object then generates the HTML code used to define a new version of the web page on the client.

The first line of the file 400 includes a directive in the format:

<%@ directive {attribute=value} %>

where *directive* may include without limitation "page", "cache", or "import". Directives are used by the page compiler when processing a dynamic content resource to determine such characteristics as buffering semantics, session state requirements, error handling schemes, scripting languages, transaction semantics, and import directions. Directives may be located anywhere within a page file.

In the second line, <html> is a standard HTML starting tag, which is passed through to the resulting HTML code as a literal (i.e., without additional processing

to render the resulting HTML code). In HTML, the <html> indicates the beginning of the HTML file and is paired with the closing tag on line 21, </html>, which is also a literal.

A code declaration block is located at lines 3-10 of the file 400. Generally, server-side code declaration blocks define page object and control object member variables and methods that are executed on the server. In the format:

<script runat = "server" [language = "*language*"][src = "*externalfile*"]>

.....

</script>

where the language and src parameters are optional. In an embodiment of the present invention, code declaration blocks are defined using <script> tags that contain a "runat" attribute having a value set to "server". Optionally, a "language" attribute may be used to specify the syntax of the inner code. The default language may represent the language configuration of the overall page; however, the "language" attribute in the code declaration block allows a developer to use different languages within the same web page implementation including, for example, Jscript and PERL (Practical Extraction and Report Language). The <script> tag may also optionally specify a "src" file, which is an external file from which code is inserted into the dynamic content resource for processing by the page compiler. It should be understood that the disclosed syntax is used in one embodiment, however, alternative embodiments may employ different syntaxes within the scope of the present invention.

In FIG. 4, two subroutines are declared in Visual Basic format within the code declaration block: AddButton_Click and DeleteButton_Click. Both subroutines take two input parameters, "Source" and "E", and are executed on the server in response to an HTTP request when a client-side click event is detected on the corresponding button. In the AddButton_Click subroutine, the text in a UserName text box is concatenated on to the word "Add" and loaded into the Text data member of Message. In the DeleteButton_Click subroutine, the text in the UserName text box is concatenated on to the word "Delete" and loaded into the Text data member of Message. Although not shown in FIG. 4, member variables of server-side control objects may be declared in the code declaration block of the file

400. For example, using a Visual Basic syntax, the key word space "DIM" declares a data variable of a server-side control object.

A "code render block" (not shown) can also be included in a dynamic content resource. In an embodiment of the present invention, a code render block executes in a single "rendering" method that executes at page render time. A code render block satisfies the following format (although other formats are contemplated in alternative embodiments):

<% *InlineCode* %>

where *InlineCode* includes self-contained code blocks or control flow blocks that execute on the server at page render time.

Inline expressions may also be used within a code render block, using the exemplary syntax:

<%= *InlineExpression* %>

where the expression contained in an *InlineExpression* block is ultimately encompassed by a call to "Response.Write(*InlineExpression*)" in a page object, which writes the value resulting from *InlineExpression* into an appropriate place holder in the declaration. For example, *InlineExpressions* may be included in a code render block as follows:

<font size = "<%=x%>" > Hi <%=Name%>, you are <%=Age%>!

which outputs a greeting and a statement about a person's age in a font stored in the value "x". The person's name and age are defined as strings in a code declaration block (not shown). The resulting HTML code is rendered at the server for transmission to the client in the HTTP response so as to include the values of the *InlineExpressions* at appropriate locations:

<font size = "12" > Hi Bob, you are 35!

On line 11 of file 400, <body> is a standard HTML tag for defining the beginning of the body of the HTML document. On line 20 of file 400, the closing tag, </body>, is also shown. Both the <body> and </body> are literals in an embodiment of the present invention.

Within the body section of the HTML file 400, on line 12, the starting tag, <form>, of an HTML form block is found in FIG. 4. The ending tag, </form>, of the form block is found on line 19 of the HTML file 400. An optional parameter "id" may also be included in the HTML control tag, <form>, to associate a given

identifier with the form block, thereby allowing multiple form blocks to be included in a single HTML file.

On line 18 of file 400, a server-side label, identified by "Message", is declared. The "Message" label is used in the code declared at lines 5 and 8 of the file 400 to display a label on the web page.

Within a form block, three exemplary HTML control tags are shown, corresponding to the user interface elements 106, 108, and 110 of FIG. 1. The first user interface element is declared on line 13 of the file 400 corresponding to a textbox. The text literal "User Name:" declares a label positioned to the left of the textbox. The input tag with type="Text" declares a textbox server-side control object having an id equaling "UserName" as a server-side control object that renders a textbox client-side user interface element. Lines 15 and 16 of file 400 declare the client-side user interface elements shown as buttons 108 and 110 of FIG. 1, respectively. Note that the "OnServerClick" parameter specifies the appropriate sub-routine declared in the code declaration block of the file 400. As such, the server-side button control objects generated in response to the declarations in file 400 render the HTML code for the client-side buttons and an associated server-side code for implementing the button click events.

The textbox and buttons declared in file 400 are examples of HTML server control declarations. By default, all HTML tags within an ASP+ resource are treated as literal text content and are programmatically inaccessible to page developers. However, page authors can indicate that an HTML tag should be parsed and treated as an accessible server control declaration by marking it with a "runat" attribute with a value set to "server". Each server-side control object may optionally be associated with a unique "id" attribute to enable programmatic referencing of the corresponding control object. Property arguments and event bindings on server-side control objects can also be specified using declarative name/value attribute pairs on the tag element (e.g., the OnServerClick equals "MyButton_Click" pair.

In an embodiment of the present invention, a general syntax for declaring HTML control objects is as follows:

<HTMLTag id = "Optional Name" runat = server>

.....

</HTMLTag>

where the *OptionalName* is a unique identifier for the server-side control object. A list of currently supported HTML tags and the associated syntax and COM+ class are illustrated in TABLE 1, although other HTML tags are contemplated within the scope of the present invention.

HTML Tag Name	Example	COM+ Class
<a>	 My Link 	AnchorButton
<img>	<img id = "MyImage" runat = server>	Image
		Label
	Some contents	Panel
<form>	<form id = "MyForm" runat = server> </form>	FormControl
<select>	<select id = "MyList" runat = server> <option>One</option> <option>Two</option> <option>Three</option> </select>	DropDownList
<input type = file>	<input id = "MyFile" type = file runat = server>	FileInput
<input type = text>	<input id = "MyTextBox" type = text>	TextBox
<input type = password>	<input id = "MyPassword" type = password>	TextBox
<input type = reset>	<input id = "MyReset" type = reset>	Button
<input type = radio>	<input id = "MyRadioButton" type = radio runat = server>	RadioButton
<input type = checkbox>	<input id = "MyCheck" type = checkbox runat = server>	CheckBox
<input type = hidden>	<input id ="MyHidden" type = hidden runat = server>	HiddenField
<input type = image>	<input type = image src = "foo.jpg" runat = server>	ImageButton
<input type = submit>	<input type = submit runat = server>	Button
<input type = button>	<input type = button runat = server>	Button
<button>	<button id = MyButton runat = server>	Button
<textarea>	<textarea id = "MyText" runat = server> This is some sample text </textarea>	TextArea

TABLE 1

In addition to standard HTML control tags, an embodiment of the present invention enables developers to create re usable components that encapsulate common programmatic functionality outside of the standard HTML tag set. These custom server-side control objects are specified using declarative tags within a page file. Custom server-side control object declarations include a "runat" attribute with a value set to "server". Optionally, the unique "id" attribute may be specified to enable programmatic referencing of the custom control object. In addition, declarative name/value attribute pairs on a tag element specify property arguments and event bindings on a server-side control object. In line template parameters may also be bound to a server-side control object by providing an appropriate "template" prefix child-element to the parent server control object. A format for a custom server-side control object declaration is:

<servercntrlclassname id="*OptionalName" [propertyname="propval"]* runat=server/>

where *servercntrlclassname* is a name of an accessible server control class, *OptionalName* is a unique identifier of the server-side control object, and *propval* represents an optional property value in the control object.

Using an alternative declaration syntax, XML tag prefixes may be used to provide a more concise notation for specifying server-side control objects within a page, using the following format:

<tagprefix:classname id = *"OptionalName"* runat = server/>

where *tagprefix* is associated with a given control name space library and *classname* represents a name of a control in the associated name space library. An optional *propertyvalue* is also supported.

In summary, an embodiment of the present invention includes server-side control objects that are created and executed on the server to generate HTML code that is sent to a client. The HTML code may embody any valid HTML constructs and may, for example, reference ACTIVEX-type controls, JAVA applets, scripts, and any other web resources to produce user interface buttons and other user interface elements at the client. A user at the client may interact with these user interface elements, which logically correspond to the server-side control objects, and send a request back to the server. The server side control objects are recreated on the server to process the data, events, and other characteristics of the user interface

elements so as to generate the next round of HTML code to be transmitted in a response to the client.

With reference to FIG. 5, an exemplary computing system for embodiments of the invention includes a general purpose computing device in the form of a conventional computer system 500, including a processor unit 502, a system memory 504, and a system bus 506 that couples various system components including the system memory 504 to the processor unit 500. The system bus 506 may be any of several types of bus structures including a memory bus or memory controller, a peripheral bus and a local bus using any of a variety of bus architectures. The system memory includes read only memory (ROM) 508 and random access memory (RAM) 510. A basic input/output system 512 (BIOS), which contains basic routines that help transfer information between elements within the computer system 500, is stored in ROM 508.

The computer system 500 further includes a hard disk drive 512 for reading from and writing to a hard disk, a magnetic disk drive 514 for reading from or writing to a removable magnetic disk 516, and an optical disk drive 518 for reading from or writing to a removable optical disk 519 such as a CD ROM, DVD, or other optical media. The hard disk drive 512, magnetic disk drive 514, and optical disk drive 518 are connected to the system bus 506 by a hard disk drive interface 520, a magnetic disk drive interface 522, and an optical drive interface 524, respectively. The drives and their associated computer-readable media provide nonvolatile storage of computer readable instructions, data structures, programs, and other data for the computer system 500.

Although the exemplary environment described herein employs a hard disk, a removable magnetic disk 516, and a removable optical disk 519, other types of computer-readable media capable of storing data can be used in the exemplary system. Examples of these other types of computer-readable mediums that can be used in the exemplary operating environment include magnetic cassettes, flash memory cards, digital video disks, Bernoulli cartridges, random access memories (RAMs), and read only memories (ROMs).

A number of program modules may be stored on the hard disk, magnetic disk 516, optical disk 519, ROM 508 or RAM 510, including an operating system 526, one or more application programs 528, other program modules 530, and program

data 532. A user may enter commands and information into the computer system 500 through input devices such as a keyboard 534 and mouse 536 or other pointing device. Examples of other input devices may include a microphone, joystick, game pad, satellite dish, and scanner. These and other input devices are often connected to the processing unit 502 through a serial port interface 540 that is coupled to the system bus 506. Nevertheless, these input devices also may be connected by other interfaces, such as a parallel port, game port, or a universal serial bus (USB). A monitor 542 or other type of display device is also connected to the system bus 506 via an interface, such as a video adapter 544. In addition to the monitor 542, computer systems typically include other peripheral output devices (not shown), such as speakers and printers.

The computer system 500 may operate in a networked environment using logical connections to one or more remote computers, such as a remote computer 546. The remote computer 546 may be a computer system, a server, a router, a network PC, a peer device or other common network node, and typically includes many or all of the elements described above relative to the computer system 500. The network connections include a local area network (LAN) 548 and a wide area network (WAN) 550. Such networking environments are commonplace in offices, enterprise-wide computer networks, intranets, and the Internet.

When used in a LAN networking environment, the computer system 500 is connected to the local network 548 through a network interface or adapter 552. When used in a WAN networking environment, the computer system 500 typically includes a modem 554 or other means for establishing communications over the wide area network 550, such as the Internet. The modem 554, which may be internal or external, is connected to the system bus 506 via the serial port interface 540. In a networked environment, program modules depicted relative to the computer system 500, or portions thereof, may be stored in the remote memory storage device. It will be appreciated that the network connections shown are exemplary, and other means of establishing a communication link between the computers may be used.

In an embodiment of the present invention, the computer 500 represents a web server, wherein the processor 502 executes a page factory module on an ASP+ resource stored on at least one of storage media 516, 512, 514, 518, 519, or memory

504. HTTP responses and requests are communicated over the LAN, 548, which is coupled to a client computer 546.

FIG. 6 illustrates a process flow diagram representing server-side processing of a page object and other control objects in an embodiment of the present invention. In operation 600, a page object constructor is called by the page factory module 308 (see FIG. 3). As a result, a page object (see e.g., the page object 314 in FIG. 3) is created to logically correspond to the web page user interface element on the client. In operation 602, the page factory module calls the ProcessRequest() member function of the page object, which initiates the staged operations for processing the HTTP request received from a client. In a first stage of one embodiment of the present invention, a server-side Create operation (not shown) creates the descendant server-side control objects contained in the control object hierarchy of the page object, that is, constructors for child control objects are recursively called to create the control objects during the lifetime of the processing of the HTTP request.

In an alternate embodiment, however, creation of child control objects is deferred until the control object is required for a given processing step (e.g., handling a postback event, handling postback data, loading or saving a viewstate, resolving data binding, or rendering HTML code for the corresponding user interface element). The latter embodiment, which is said to implement "deferred control object creation", is an optimization that can alleviate unnecessary CPU and memory utilization. For example, a user input event received from the client may result in the creation of a completely different web page. In this case, it is unnecessary to instantiate an entire control object hierarchy of the previous page only to process an event that immediately results in the termination of the control object hierarchy and the instantiation of a new and different control object hierarchy for a new page.

In response to the server call to the page object's ProcessRequest method, operations 604 through 620 may be executed by the page object and by individual descendant control objects, depending in part on the data of a given HTTP request. In an embodiment of the present invention, the operation 604-620 are performed for each individual object in the order illustrated in FIG. 6; however, a given operation for one object may occur out of order or not at all with respect to a given operation of another object, depending on the HTTP request. For example, a first object may

perform its Init operation 604 and its Load operation 606, and begin postback data processing operation 608, before a descendant control object performs its own Init operation 604 and Load operation 606 by virtue of deferred control object creation. The order of operation processing by the page object and descendent control objects depends on various factors, including without limitation the nature of the data in the HTTP request, the configuration of the control object hierarchy, the current state of the control objects, and whether deferred control object creation is implemented.

The Init operation 604 initializes a control object after it is created by executing any server-side code associated with initialization in the dynamic content resource. In this manner, each server-side control object may be customized with specific server-side functionality that is declared in the dynamic content resource. In an embodiment of the present invention, dynamic content code intended to customize or extend the base page control classes as declared by the page developer in the ASP+ resource on the server. When the ASP+ resource is compiled, the declared code is included in the appropriate initialization code (e.g., the Init() methods of the page object and the descendent control objects). The Init operation 604 executes this code to customize or extend the page base class and the base classes for descendent control objects.

In an embodiment of the present invention, state management of the server-side control objects is supported in a Load operation 606 and a Save operation 616, which use a transportable state structure to accommodate the stateless model for client server systems by restoring server-side control objects to their previous states. In one embodiment, the state is communicated to and from the server in one or more hidden HTML fields of an HTTP request/response pair, although other transportable state structures are contemplated within the scope of the present invention.

In a given sequence of requests and responses relating to the current page between a client and a server, the states of one or more control objects are recorded into a transportable state structure by the Save operation 616 after the processing of a previous request. In an embodiment of the present invention, additional state information is also included in the transportable state structure, including hierarchical information or control object identifiers to allow the server to associate a given state with the appropriate control object. In a subsequent HTTP request, the state information is returned to the server in the transportable state structure. The

server extracts the state information from the received transportable state structure and loads the state data into the appropriate control objects within the control object hierarchy to restore each control object to its state as it existed prior to a previous HTTP response. After the current request is processed, the states of one or more server-side control objects are again recorded into the transportable state structure by the Save operation 616, and the transportable state structure is returned to the client in the next HTTP response.

As a result of the Load operation 606, each server-side control object is placed in a state consistent with its state prior to a previous HTTP response. For example, if a text box control object includes a property value equaling "JDoe" prior to a previous HTTP response, the Load operation 606 restores the same control object to its previous state, in part by loading the text string "JDoe" into the property value. In addition, whether the state of a given object is stored and restored is configurable.

In summary of one embodiment of the present invention, the state of one or more server-side control objects is "saved" after processing. The saved state information is transmitted to the client in a response. The client returns the saved state information to the server in a subsequent response. The server loads the state information a freshly instantiated server-side control object hierarchy, such that the state of the hierarchy is restored to its previous state.

An alternative embodiment may maintain the state information on the server or at some other web location accessible by the server during the round trip from the server to the client, and then back to the server. After the client request is received by the server, this state information may be retrieved by the server and loaded into the appropriate server-side control object(s) in the control object hierarchy.

In operation 608, postback data received from the HTTP request is processed. Postback data may be included in the payload of the HTTP request in key-value pairs, in a hierarchical representation (e.g., XML), or in other data representations, such as RDF ("Resource Description Framework"). Operation 608 parses the payload to identify a unique identifier of a server-side control object. If the identifier (e.g. "page1:text1") is found and the identified server-side control object exists in the control object hierarchy, the corresponding postback data is passed to the control object. For example, referring to FIG. 1, a unique identifier

associated with textbox 106 and the text "JDoe" are communicated in the payload of the HTTP request 114 to the web server 116. Operation 608 parses the payload of the HTTP request 114 and obtains the unique identifier of the textbox 106 and its associated value (i.e., "JDoe"). Operation 608 then resolves the unique identifier of the textbox 106 to identify the corresponding server-side control object and passes the "JDoe" value to the target object for processing.

As discussed with regard to the Load operation 606, the property values of server-side control objects may be restored to their previous states. In response to the receipt of postback data, the server side control object determines whether the passed-in postback value causes a change from the corresponding property's previous value. If so, the change is logged in a change list to indicate a data change for the associated control object. After all postback data has been processed within the control object hierarchy, a call may be made to a control object method to raise one or more postback data changed events to one or more non-user-interface server components, such as a stock price look-up application running on the server. An example of a postback data changed event is an event indicating that postback data has caused a property of a server-side control object to change. In an exemplary embodiment, such an event can be sent to a system-provided event queue so that server-side code, which may be registered to process the event, can be invoked. The server-side code may then call a method of the non-user-interface server component. In this manner, a server-side non-user-interface server component can respond to events triggered by a change in data of a server-side control object. Alternative methods of implementing events are also contemplated in the scope of the present invention, including using application-provided event queues, polling, and processing interrupts.

In operation 610, postback events are handled. Postback events may be communicated in the payload of the HTTP request. Operation 610 parses a specified event target (e.g., labeled "_EVENTTARGET" in an embodiment of the present invention) identifying the server-side control object to which the event is directed. Furthermore, operation 610 parses the located event arguments, if any, and provides the event argument (e.g., labeled "_EVENTARGUMENT" in an embodiment of the present invention) to the specified server-side control object. The control object raises its events for processing by server-side code, which calls a

method of a non-user-interface server component (e.g., a server-side stock price look-up application) associated with the dynamic content resource.

Operation 612 resolves data binding relationships between the server-side control objects and one or more databases accessible by the server, thereby updating in control object properties with database values and/or updating database fields with values of control object properties. In an embodiment of the present invention, properties of server-side control objects may be associated (or data bound) to properties of a parent data binding container, such as a table in a server-side application database. During the data binding operation 612, the page framework may update a data bound control object property with the value of the corresponding parent data binding container property. In this manner, user interface elements on the web page of the next response accurately reflect updated property values, because the control object properties to which the user interface elements correspond have been automatically updated during the data binding operation 612. Likewise, control object properties can also be updated to the parent data binding container fields, thereby updating a server-side application database with postback input from a server-side control object.

Operation 614 performs miscellaneous update operations that may be executed before the control object state is saved and the output is rendered. Operation 616 requests state information (i.e., viewstate) from one or more control objects in the control object hierarchy and stores the state information for insertion into a transportable state structure that is communicated to the client in the HTTP response payload. For example, a "grid" control object may save a current index page of a list of values so that the "grid" control object may be restored to this state after a subsequent HTTP request (i.e., in operation 606). As described above, the viewstate information represents the state of the control object hierarchy prior to any subsequent actions by the client. When the viewstate information is returned, it will be used to place the control object hierarchy in that previous state prior to processing any client postback input or databinding.

The render operation 618 generates the appropriate authoring language output (e.g., HTML data) for communication to the client in an HTTP response. Rendering is accomplished through a top-down hierarchical tree walk of all server-side control objects and embedded rendering code. Operation 620 performs

any final cleanup work (e.g., closing files or database connections) before the control object hierarchy is terminated. Processing then returns to operation 602 and proceeds to operation 622 where the page object is terminated by calling its destructor.

FIG. 7 illustrates a representation of an exemplary server side Control class in an embodiment of the present invention. The server-side control class defines the methods, properties and events common to all server-side control objects in an embodiment of the present invention. More specific Control classes (e.g., a server-side button control object that corresponds to a client-side button in a web page) are derived from the control class. A Control class 700 is illustrated as including memory that stores properties 702 and methods 704. Other Control class embodiments having a different combination of data members and methods are also contemplated within the scope of the present invention.

In the illustrated embodiment, properties 702 are public. Property "ID" is a readable and writable string value indicating a control object identifier. Property "Visible" is a readable and writable Boolean value that indicates whether the authoring language data for a corresponding client-side user interface element should be rendered. Property "MaintainState" is a readable and writable Boolean value that indicates whether the control object should save its viewstate (and the viewstate of its children) at the end of the current page request (i.e., in response to a save operation 616 in FIG. 6). Property "Parent" is a readable reference to a ControlContainer (see FIG. 8) associated with the current control object in the control object hierarchy. Property "Page" is a readable reference to the root page object in which the current control object is hosted. Property "Form" is a readable reference to a FormControl object in which the current control object is hosted. Property "Trace" is a readable reference that allows a developer to write a trace log. PropertyBindingContainer is a readable reference to the control object's immediate data binding container. Property "Bindings" is a readable reference to a collection of the control object's data binding associations.

Methods 704 include methods for processing requests and accessing data members of the control object. In one embodiment, the methods are referenced by pointers stored in the memory space of the control object. This referencing means is also employed in embodiments of other server-side objects, including a container

control object, a control collection object, and a page object. Method "Init()" is used to initialize child control objects after they are created (see operation 604 of FIG. 6). Method "Load()" is used to restored the viewstate information from a previous HTTP request (see operation 606 of FIG. 6). Method "Save()" is used to save viewstate information for use with a later HTTP request (see operation 616 of FIG. 6). Method "PreRender()" is used to perform any pre-rendering steps necessary prior to saving viewstate and rendering content (see operation 614 of FIG. 6). Method "Render (TextWriter output)" is used to output authoring language code for the user interface element corresponding to the current control object (see operation 618 of FIG. 6). The code is communicated through the output stream (passed to it in the "output" parameter) in order to store the code in the response to the client. Method "Dispose()" is used to perform final clean-up work before terminating the control object (see operation 620 of FIG. 6).

Method "GetUniqueID()" obtains a unique, hierarchically qualified, string identifier for the current control object. Method "GetControlWithID(String id)" returns a reference to an immediate child control object with the provided identifier ("id"). Method "GetControlWithUniqueID(String id)" returns a reference to a child control object having a unique hierarchical identifier ("id").

Method "PushControlPropertyTwoBindingContainer (String prop Name)" is used to update a binding container for two-way data binding from post-back data when the post-back data value changes within the server-side control object. Method "PushBindingContainerPropertyTwoControl(String prop Name)" is used to update a server-side control object property with a current binding container value. Method "HasBindings()" returns of Boolean value indicating whether a control object has any binding associations. These three functions are used to resolve binding relationships between properties of server-side control objects and attributes in server-side datastores (see the databinding operation 612 of FIG. 6).

FIG. 8 illustrates an exemplary server-side ContainerControl class in an embodiment of the present invention. The ContainerControl class provides an implementation that supports nested child control objects, and automatically serializes and de-serializes viewstate information into a transportable state structure. A ContainerControl class 800 includes memory that stores properties 802 and methods 804. Property "Controls" is a readable reference to a "ControlCollection"

of the control objects children in the control object hierarchy. (See FIG. 9.) Property "StateCollection" is a readable reference to a dictionary of viewstate information used to maintain the state of a control object across multiple page requests. Method "HasControls()" returns a Boolean value indicating whether the control object has any child control objects.

In an alternative embodiment, the member properties and methods of the ContainerControl class 800 may be combined into the Control class 700 of FIG. 7, such that each control object is capable in and of itself to support children. If a control object of such an embodiment includes no such child objects, however, the Controls member (of type ControlCollection) would be empty (i.e., have no child objects).

FIG. 9 illustrates an exemplary server-side ControlCollection class in an embodiment of the present invention. A ControlCollection class 900 includes memory for storing properties 902 and methods 904. Property "All" is a readable and writable snapshot array of all child control objects of the current control object, ordered by an index. Property "this [index]" is a readable reference to an ordinal indexed control object within the control collection. Property "Count" is a readable value indicating the number of child control objects in the collection.

Method "Add(Control child)" is used to add a specified control object to the current collection. Method "IndexOf(Control child)" returns the ordinal index of the specified child control object in the collection. Method "GetEnumerator(bool AllowRemove)" returns the enumerator of all child control objects within the collection. Method "Remove(Control value)" removes the specified control object from the current collection. Method "Remove(int index)" removes a specified control object from the current collection based on the provided index. Method "Clear()" removes all control objects from the current collection.

FIG. 10 illustrates an exemplary server-side Page object in an embodiment of the present invention. A page base class defines the methods, properties and events common to all server-manipulated pages within an embodiment of the present invention. A page object 1000 includes memory storing properties 1002 and methods 1004. Property "ErrorPage" is a readable and writable String that is rendered in the event of an unhandled page exception. In this manner, a page object returns a readable error page to the client in an HTTP response. Property

"Requests" is a readable reference to an HTTPRequest, which is provided by the web server framework that provides functionality for accessing incoming HTTP request data. Property "Response" is a readable reference to an HTTP response, provided by the web server framework, which provides functionality for transmitting HTTP response data to a client. Property "Application" is a readable reference to an HTTPApplication, provided by the web server framework. Property "Session" is a readable reference to an HTTPSession, provided by the web server framework. Property "Server" is a readable reference to a ServerObject, which is an application server page-compatible utility object. Property "Context" is a readable reference to all of the web server objects provided by the web server framework, which allows a developer to gain access to additional pipeline-module exposed objects. Property "IsFirstLoad" is a readable Boolean value that indicates whether the page object is being loaded and accessed for the first time or in response to a client postback request.

Method "Load()" is used to initialize the page object and restore viewstate information from the previous page request. Method "Save()" is used to save viewstate information for use with a later page request. Method "HandleError(Exception errorInfo)" is used to handle an unhandled error occurring during a page execution. In this event, the base class implementation redirects the client to a URL having a default error web page. Method "GetPostbackScript(Control target, String name, String arg)" returns a client-side script method associated with a given control object. Method "RegisterClientScriptBlock(String key, String script)" is used to eliminate duplicate blocks of client-side script code being sent to the client. Duplicate blocks are scripts with the same key value. Method "IHTTPHandler.ProcessRequest(HTTPContext Context)" is used to process web requests. The IHTTPHandler.ProcessRequest is called to initiate the processing of an HTTP request received from a client (see operation 602 of FIG. 6). Method "IHTTPHandler.IsReusable()" indicates whether a page object can be reused to service multiple web requests.

The embodiments of the invention described herein are implemented as logical steps in one or more computer systems. The logical operations of the present invention are implemented (1) as a sequence of processor-implemented steps executing in one or more computer systems and (2) as interconnected machine

modules within one or more computer systems. The implementation is a matter of choice, dependent on the performance requirements of the computer system implementing the invention. Accordingly, the logical operations making up the embodiments of the invention described herein are referred to variously as operations, steps, objects, or modules.

The above specification, examples and data provide a complete description of the manufacture and use of the composition of the invention. Since many embodiments of the invention can be made without departing from the spirit and scope of the invention, the invention resides in the claims hereinafter appended.

4 Brief Description of Drawings

FIG. 1 illustrates a web server for dynamically generating web page content for display on a client in an embodiment of the present invention.

FIG. 2 illustrates a flow diagram of operations for processing and rendering client-side user interface elements using server-side control objects in an embodiment of the present invention.

FIG. 3 illustrates exemplary modules in a web server used in an embodiment of the present invention.

FIG. 4 illustrates an exemplary dynamic content resource (e.g., an ASP+ resource) in an embodiment of the present invention.

FIG. 5 illustrates an exemplary system useful for implementing an embodiment of the present invention.

FIG. 6 illustrates a process flow diagram representing processing of a page object in an embodiment of the present invention.

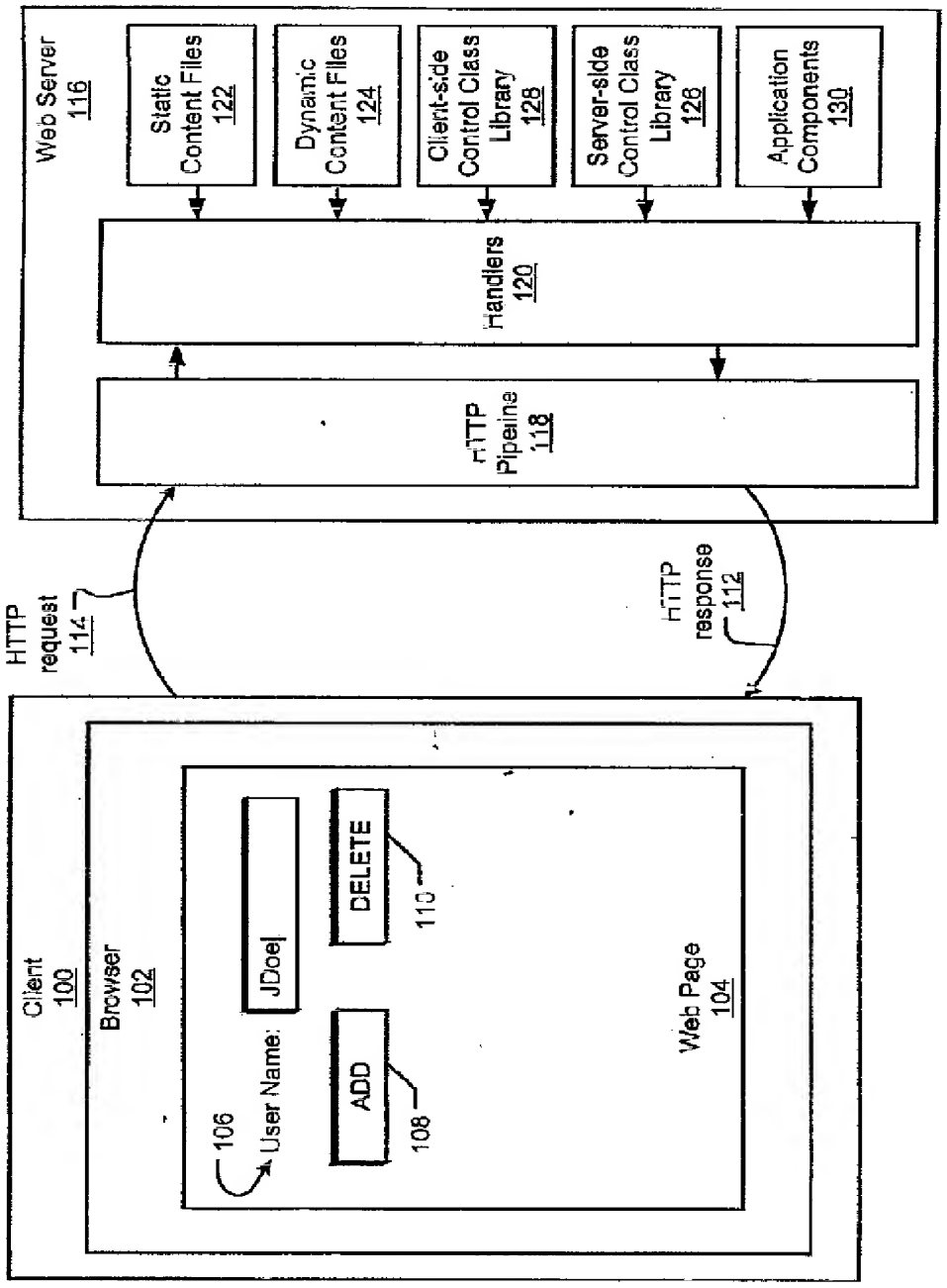
FIG. 7 illustrates an exemplary server-side Control class in an embodiment of the present invention.

FIG. 8 illustrates an exemplary server-side ContainerControl class in an embodiment of the present invention.

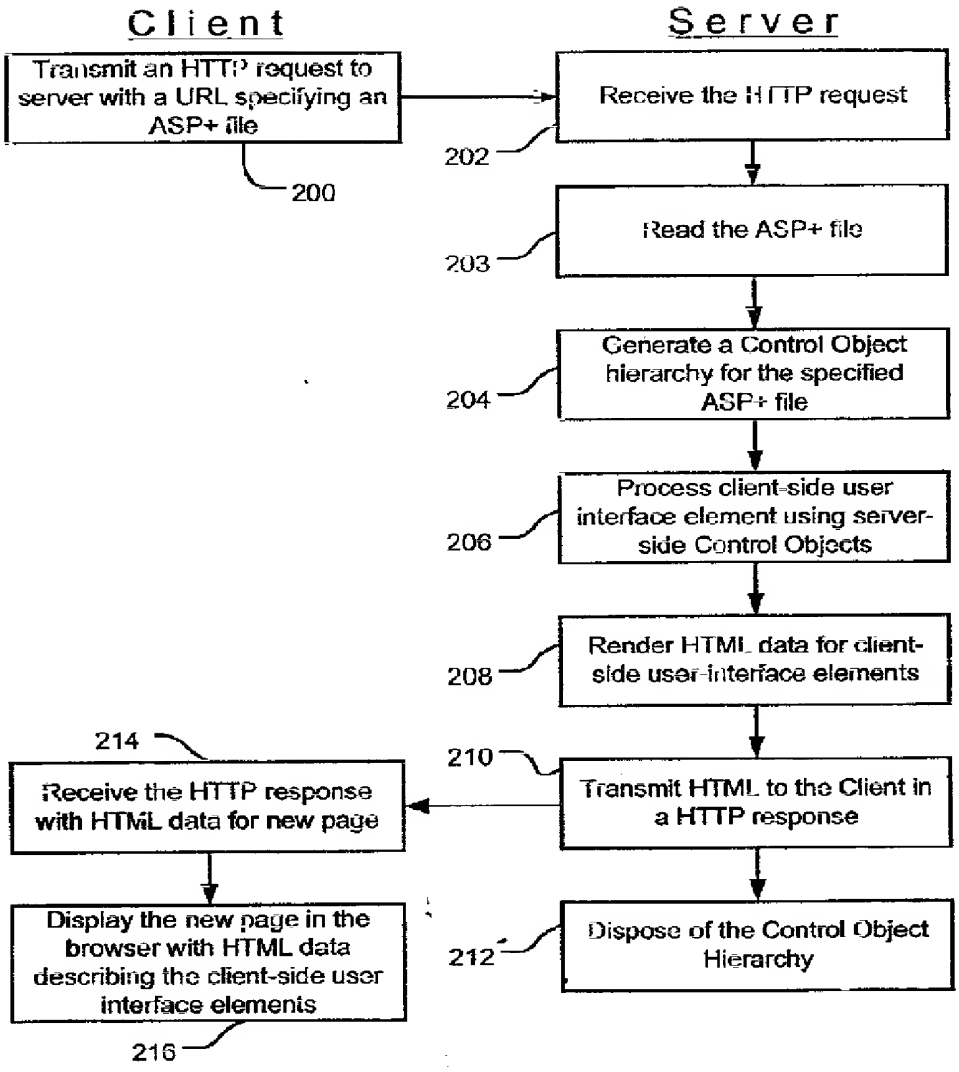
FIG. 9 illustrates an exemplary server-side ControlCollection class in an embodiment of the present invention.

FIG. 10 illustrates an exemplary server-side Page class in an embodiment of the present invention.

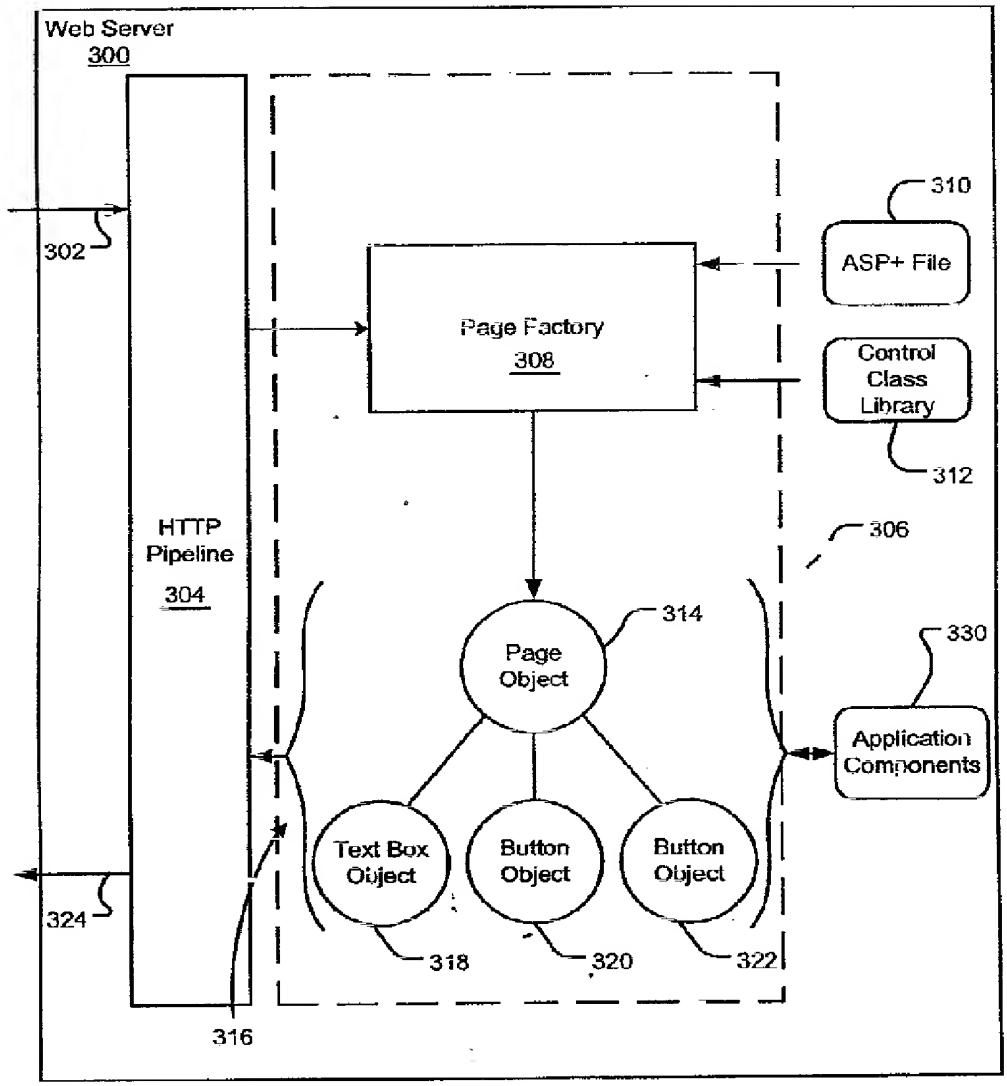
【図1】


Web Server
116
Static Content Files
122
Dynamic Content Files
124
Client-side Control Class Library
128
Server-side Control Class Library
128
Application Components
130
Handlers
120
HTTP Pipeline
118
HTTP request
114
HTTP response
112
Client
100
Browser
102
User Name:
JDoe
106
ADD
108
DELETE
110
Web Page
104

【図2】


Client
Server
Transmit an HTTP request to server with a URL specifying an ASP+ file
200
Receive the HTTP request
202
Read the ASP+ file
203
Generate a Control Object hierarchy for the specified ASP+ file
204
Process client-side user interface element using server-side Control Objects
206
Render HTML data for client-side user-interface elements
208
Transmit HTML to the Client in a HTTP response
210
Dispose of the Control Object Hierarchy
212
214
Receive the HTTP response with HTML data for new page
Display the new page in the browser with HTML data describing the client-side user interface elements
216

【図３】

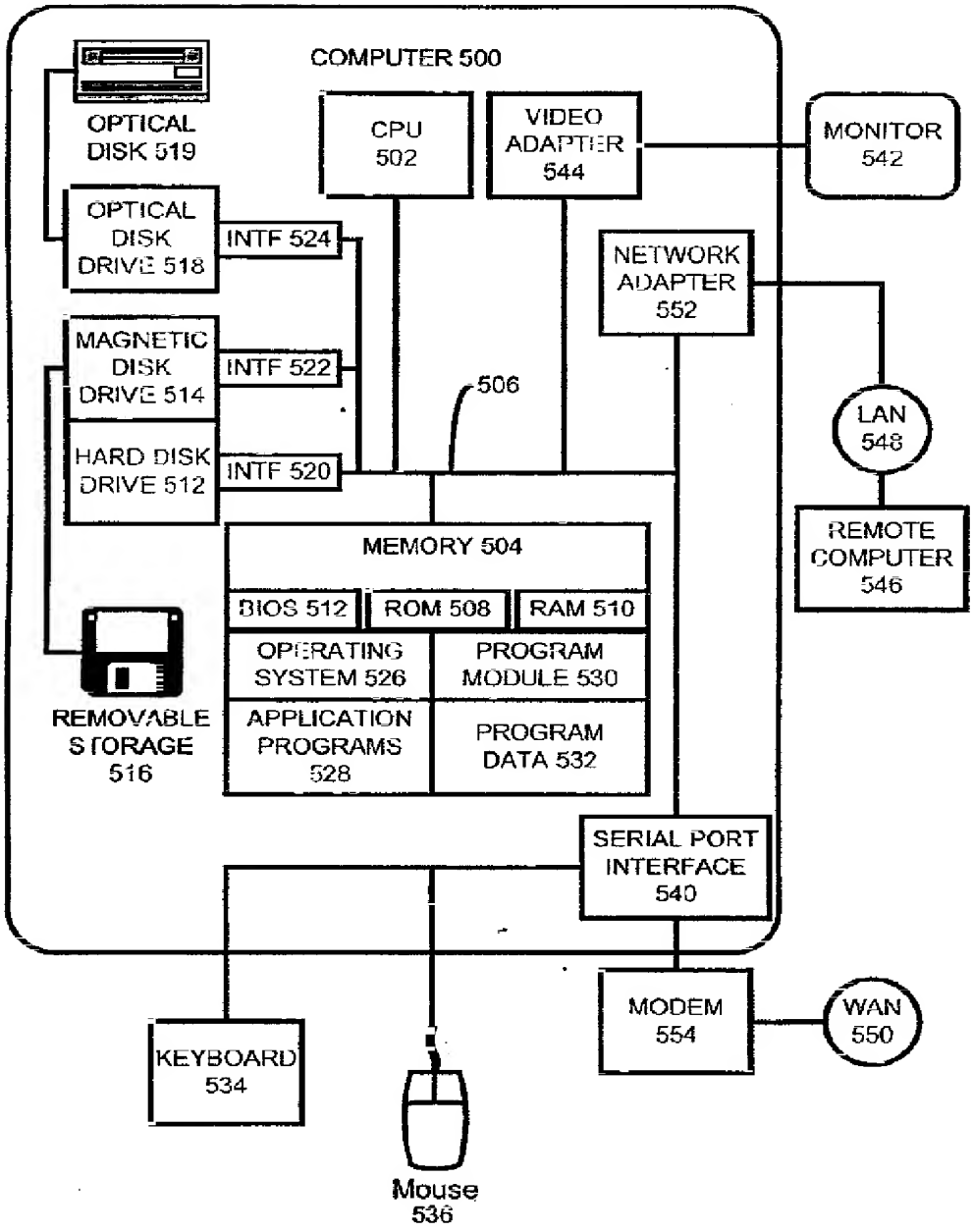

Web Server
300
302
HTTP
Pipeline
304
Page Factory
308
310
ASP+ File
Control
Class
Library
312
306
314
Page
Object
330
Application
Components
Text Box
Object
Button
Object
Button
Object
318
320
322
324
316

【図4】

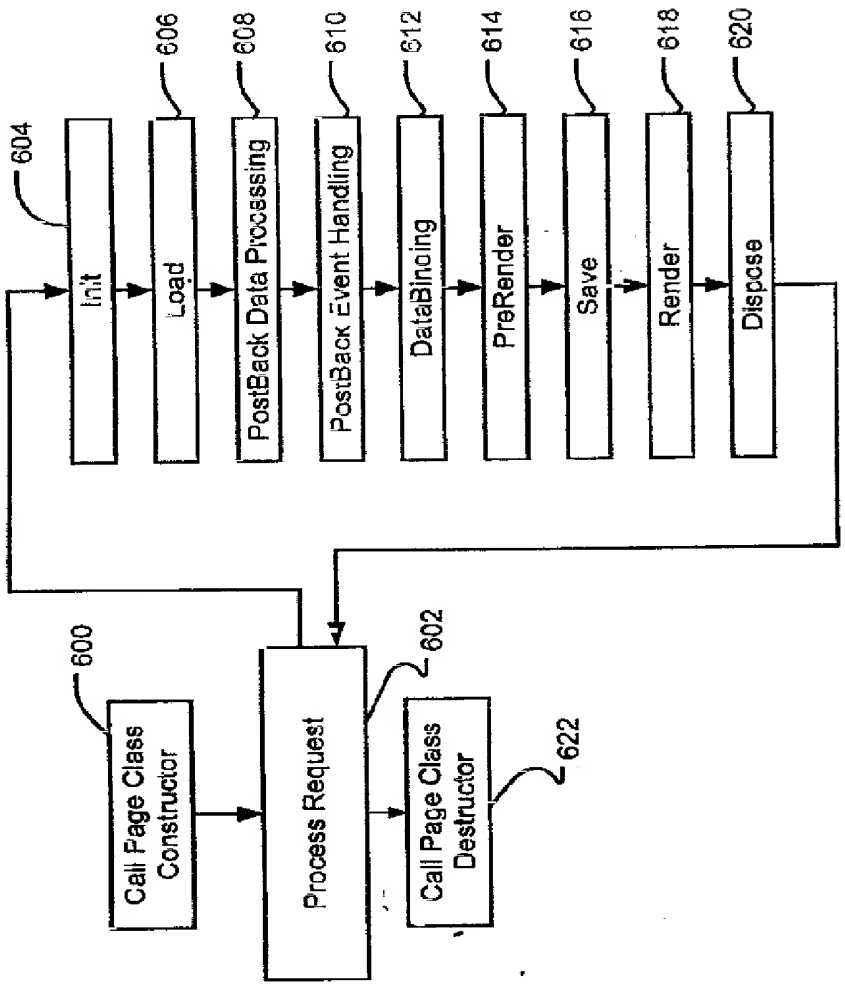
```
1   <%@ Page Language="VB" Description="Simple Sample Page" Errorpage="ErrorPage.aspx" %>
2   <html>
3       <script runat=server>
4           Sub AddButton_Click(ByVal Source as Object, By Val E as Event Args)
5               Message.Text = "Add" & UserName.Text
6           End Sub
7
8           Sub DeleteButton_Click(ByVal Source as Object, By Val E as Event Args)
9               Message.Text = "Delete" & UserName.Text
10          End Sub
        </script>
11      <body>
12          <form runat="server">
13              User Name:      <input type="Text" id="UserName" runat=server>
14              <br>
15              <button id="AddButton" value="ADD" OnClick="AddButton_Click" runat=server>
16              <button id="DeleteButton" value="DELETE" OnClick="DeleteButton_Click" runat=server>
17              <br><br>
18              <span id="Message" runat=server> </span>
19          </form>
20      </body>
21  </html>
```

400

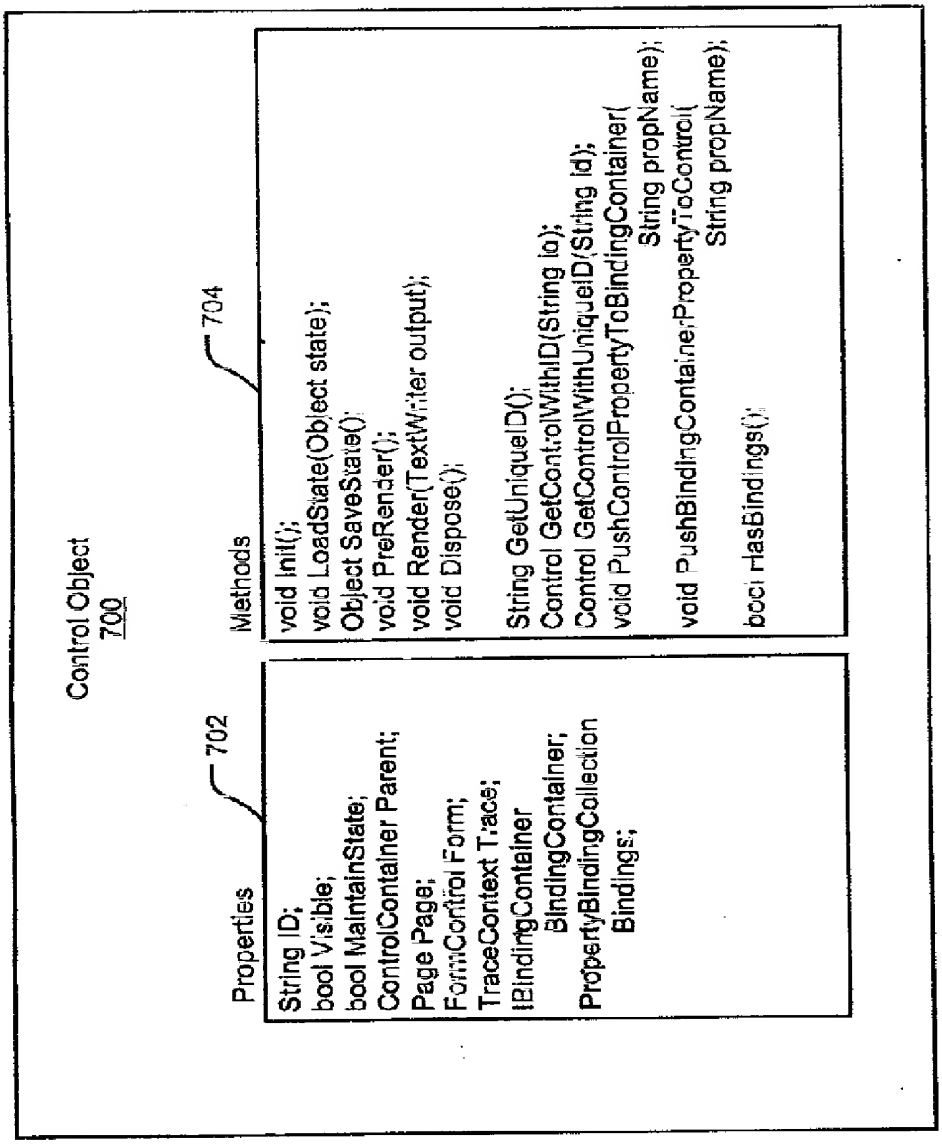
【図5】


COMPUTER 500
OPTICAL DISK 519
CPU 502
VIDEO ADAPTER 544
MONITOR 542
OPTICAL DISK DRIVE 518
INTF 524
NETWORK ADAPTER 552
MAGNETIC DISK DRIVE 514
INTF 522
506
LAN 548
HARD DISK DRIVE 512
INTF 520
REMOTE COMPUTER 546
MEMORY 504
BIOS 512
ROM 508
RAM 510
OPERATING SYSTEM 526
PROGRAM MODULE 530
REMOVABLE STORAGE 516
APPLICATION PROGRAMS 528
PROGRAM DATA 532
SERIAL PORT INTERFACE 540
MODEM 554
WAN 550
KEYBOARD 534
Mouse 536

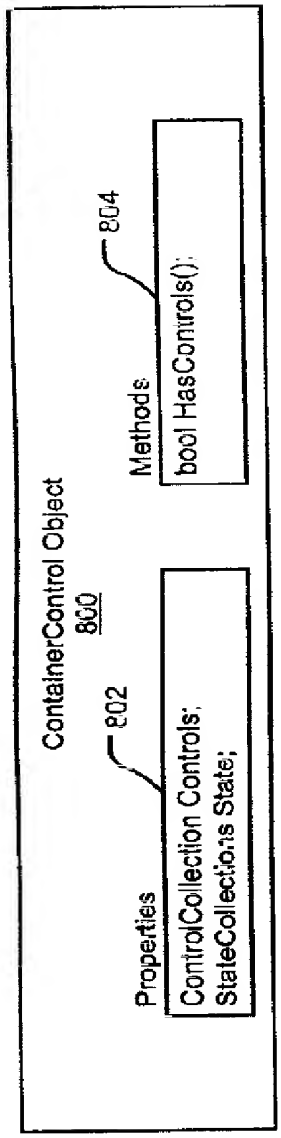
【図6】


604
606
608
610
612
614
616
618
620
Init
Load
PostBack Data Processing
PostBack Event Handling
DataBinding
PreRender
Save
Render
Dispose
600
602
622
Call Page Class Constructor
Process Request
Call Page Class Destructor

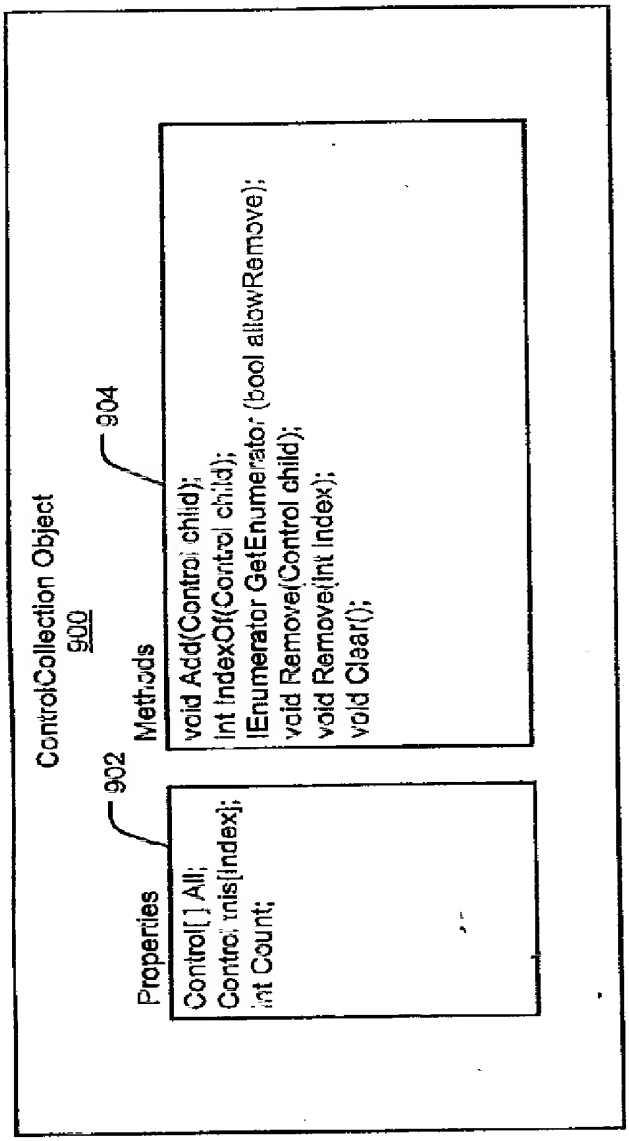
【図7】


Control Object
700
Properties
702
String ID;
bool Visible;
bool MaintainState;
ControlContainer Parent;
Page Page;
FormControl Form;
TraceContext Trace;
IBindingContainer
BindingContainer;
PropertyBindingCollection
Bindings;
Methods
704
void Init();
void LoadState(Object state);
Object SaveState();
void PreRender();
void Render(TextWriter output);
void Dispose();
String GetUniqueID();
Control GetControlWithID(String id);
Control GetControlWithUniqueID(String id);
void PushControlPropertyToBindingContainer(
String propName);
void PushBindingContainerPropertyToControl(
String propName);
bool HasBindings();

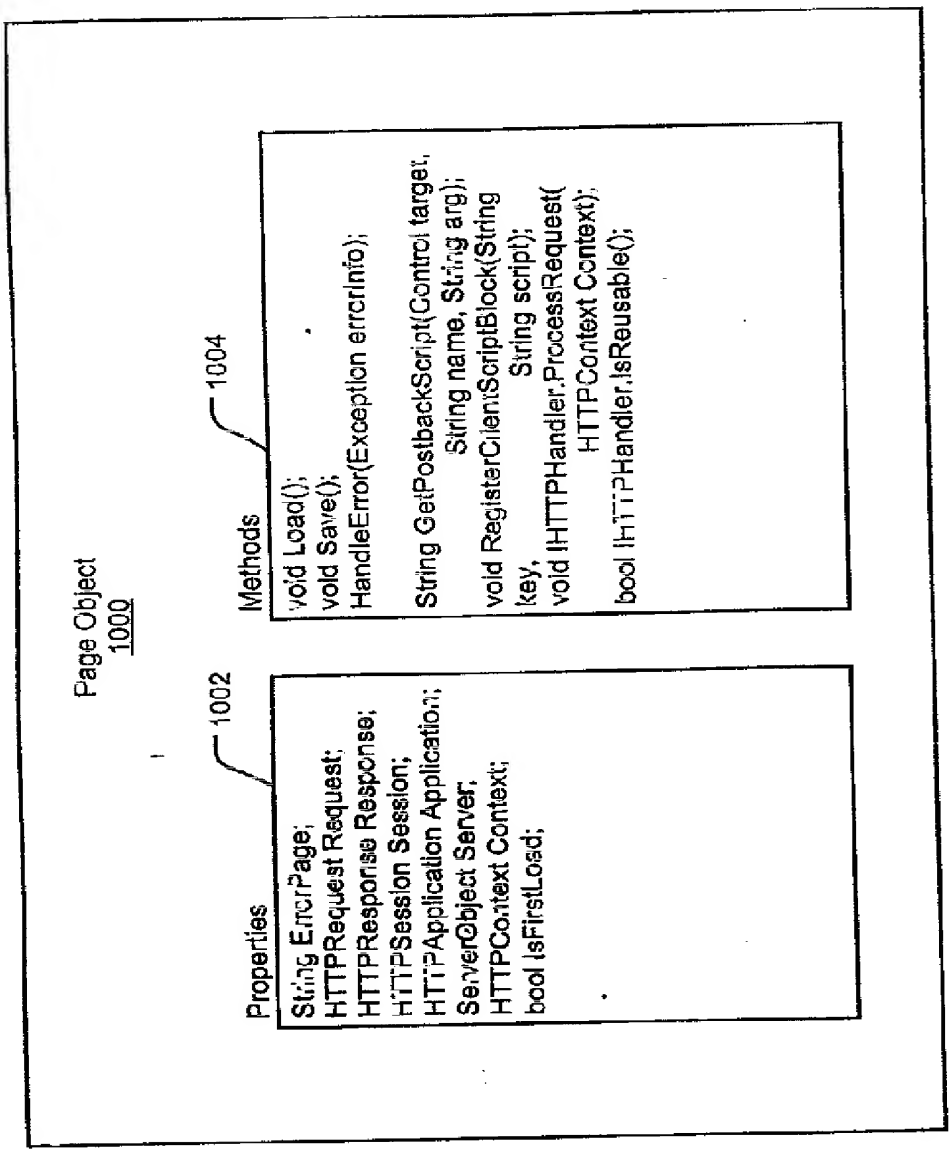
【図8】


ContainerControl Object
800
802
Properties
ControlCollection Controls;
StateCollections State;
804
Methods
bool HasControls();

【図9】


ControlCollection Object
900
902
Properties
Control[] All;
Control this[Index];
int Count;
904
Methods
void Add(Control child);
int IndexOf(Control child);
IEnumerator GetEnumerator (bool allowRemove);
void Remove(Control child);
void Remove(int Index);
void Clear();

【図10】


Page Object
1000
1002
Properties
String ErrorPage;
HTTPRequest Request;
HTTPResponse Response;
HTTPSession Session;
HTTPApplication Application;
ServerObject Server;
HTTPContext Context;
bool IsFirstLoad;
1004
Methods
void Load();
void Save();
HandleError(Exception errorInfo);
String GetPostbackScript(Control target,
String name, String arg);
void RegisterClientScriptBlock(String
key, String script);
void IHTTPHandler.ProcessRequest(
HTTPContext Context);
bool IHTTPHandler.IsReusable();

A server side control object processes and generates a client-side user interface element for display on a web page. Multiple server- side control objects may be combined into a hierarchy of server-side control objects that cooperate to generate the resulting authoring language code, such as HTML, for display of a web page on a client. The operation of processing the client-side user interface element may include at least one of an event handling operation, a postback data handling operation, a data binding operation, and a state management operation. The state management operation relates to the state of a server-side control object.

2 Representative Drawing

Fig. 2